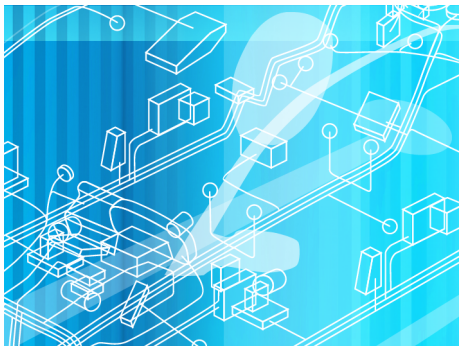
Fraunhofer Institut Software- und Systemtechnik

Fallstudie »Condition-Based Service«

Modelle für die Bewertung von logischen Architekturen und Softwarearchitekturen

Martin Große-Rhode,
Ekkart Kleinod, Stefan Mann



ISST-Bericht 84/07
Oktober 2007

Herausgeber: Fraunhofer-Gesellschaft e. V.
Institut für Software- und Systemtechnik
Leitung: Prof. Dr. Jakob Rehof
Institutsteil Berlin: Mollstraße 1
10178 Berlin
Institutsteil Dortmund: Joseph-von-Fraunhofer-Straße 20
44227 Dortmund

ISSN 0943-1624

Das Projekt VEIA

Das Projekt »Verteilte Entwicklung und Integration von Automotive-Produktlinien« wird vom Bundesministerium für Bildung und Forschung im Rahmen der Forschungsoffensive »Software Engineering 2006« unter dem Förderkennzeichen »01ISF15A« gefördert.

Am Projekt sind vier Partner beteiligt:

- BMW Group
- Fraunhofer ISST
- PROSTEP IMP GmbH
- Technische Universität München

Webseite: http://veia.isst.fraunhofer.de/

Autoren

Dieses Dokument wurde geschrieben von:

Martin Große-Rhode, Ekkart Kleinod, Stefan Mann (Fraunhofer ISST).

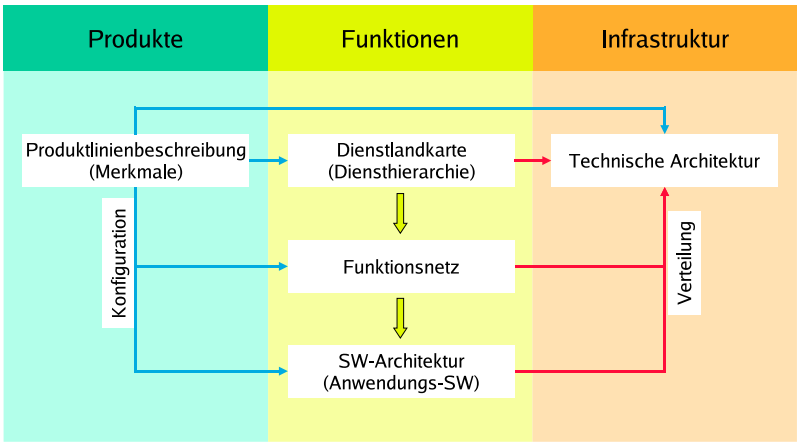
Inhalt

1	**Einleitung**	3
2	**Fallstudie**	4
3	**Produktlinienbeschreibung**	5
4	**Logische Architektur**	9
4.1	Funktionsnetz CBS	9
4.2	CBS-Umfang Bremsbelag	12
4.3	CBS-Umfang Motoröl	15
4.4	CBS-Umfang Zündkerzen	16
4.5	CBS-Umfang Gesetzliche Fahrzeuguntersuchung	18
5	**Softwarearchitekturen**	22
5.1	Produktgenerische Softwarearchitektur und Softwarekomponenten	23
5.2	Produktspezifische Softwarearchitekturen und Softwarekomponenten	25
6	**Anwendung von Metriken auf die CBS-Modellierung**	40
6.1	Produkte	40
6.2	Anwendung auf die logische Architektur von CBS	41
6.3	Anwendung auf die CBS-Softwarearchitekturen	45
7	**Zusammenfassung**	56
Literatur		57

1 Einleitung

Dieses Dokument dient der Erfassung der Fallstudie »*Condition-Based Service*« (CBS) zur Evaluation der Methoden und Konzepte, die im Rahmen des Projekts VEIA erarbeitet werden. Ausgangsbasis der Betrachtungen sind der in [GEKM07] beschriebene VEIA-Referenzprozess mit seinen identifizierten Entwicklungsartefakten. Der Referenzprozess ist in Abbildung 1 dargestellt.

Abbildung 1 Der VEIA-Referenzprozess mit den Entwicklungsartefakten.



Im vorliegenden Dokument wird die Fallstudie *CBS* (siehe Kapitel 2) beispielhaft gemäß des VEIA-Referenzprozesses modelliert: von der Produktlinienbeschreibung (Kapitel 3) über die logische Architektur (Kapitel 4) bis zu Softwarearchitekturen (Kapitel 5).[1] Die Modelle dienen in der dritten VEIA-Projektphase zur Untersuchung des Themenfelds Architekturbewertung von Produktlinien mit Hilfe von Metriken, siehe hierzu [GKM07]. Die beispielhafte Anwendung der Metriken auf Basis von Prozessfamilienpunkten (PFP) zur Bewertung von Lösungsentwürfen von Produktlinien ist in Kapitel 6 dargestellt.

[1] Die im VEIA-Referenzprozess ebenfalls vorgesehene Dienstlandkarte wird im Rahmen dieses Dokuments nicht beschrieben, siehe hierzu bspw. [GEKM07]. Auch die Beschreibung der technischen Architektur (Hardwaretopologie) ist hier nicht angegeben.

2 Fallstudie

Mit »*Condition-Based Service*« wird ein Wartungskonzept für Fahrzeuge bezeichnet, das für den Nutzer eines Fahrzeugs optimierte Servicetermine anbietet. So soll dem Fahrzeugnutzer mitgeteilt werden, wann aus Sicht des Fahrzeugs der nächste Werkstattbesuch ansteht. Die Bestimmung eines optimalen Servicetermins wird dabei am tatsächlichen Verschleiß relevanter Fahrzeugteile (z. B. Bremsbeläge, Motoröl, Zündkerzen etc.) ausgerichtet. Dieses Wartungskonzept wird durch Software im E/E-System des Fahrzeugs unter Zuhilfenahme von bereits im Fahrzeug erhobenen Sensordaten realisiert.

Abbildung 2 Illustration der Funktionalität »Condition-Based Service« (CBS).

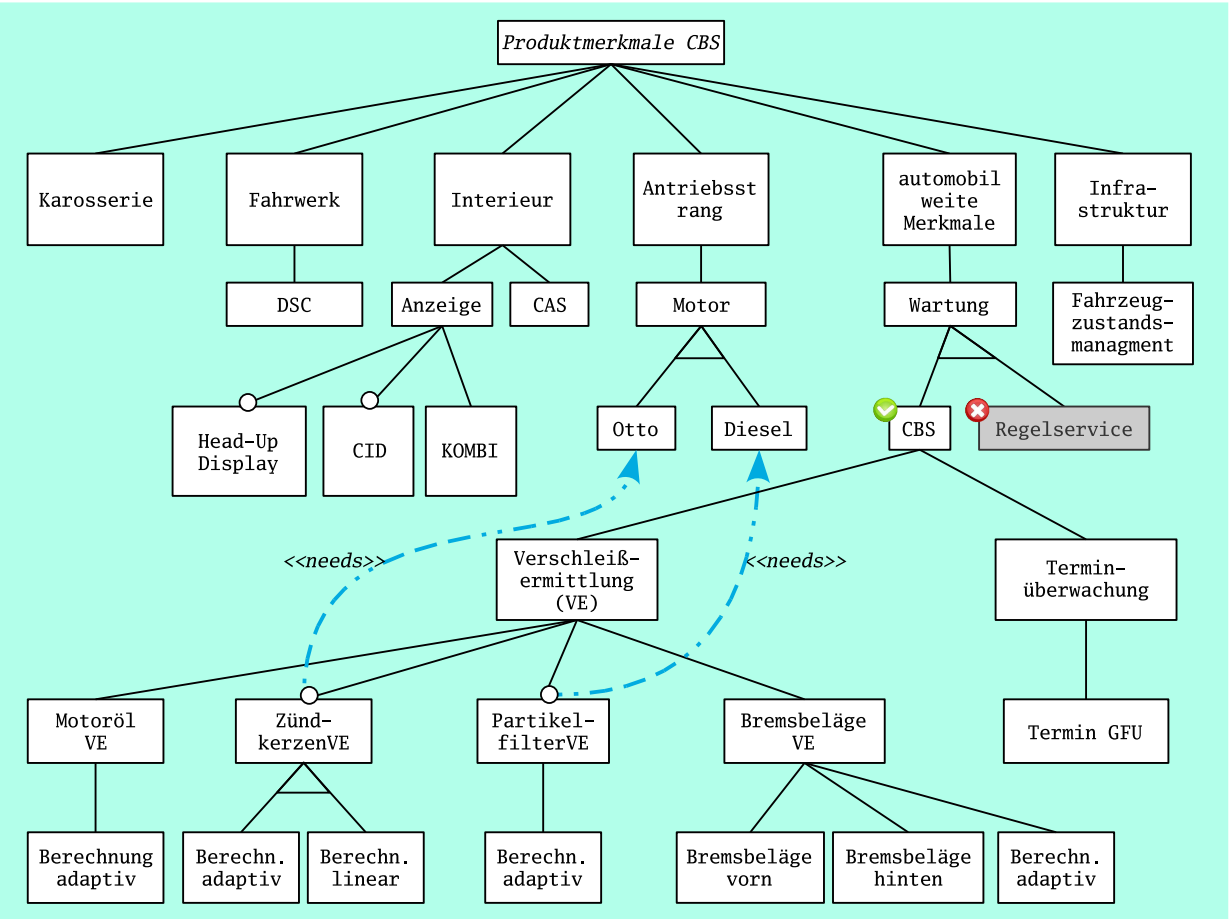
Abbildung 2 illustriert, wie sich die Funktionalität *CBS* dem Nutzer eines Fahrzeugs darstellt.[2]

[2] Quellenangabe: Abbildung 2 wurde der Präsentation von Simon Euringer (BMW Group) auf der 2. Steuerkreissitzung zum Projekt VEIA, 11. Mai 2007 in München entnommen.

3 Produktlinienbeschreibung

Die Produktlinienbeschreibung definiert auf Anforderungsebene die funktionalen Merkmale von *CBS* sowie relevante Merkmale aus der »Systemumgebung« von *CBS*. Die auf dieser Ebene vorliegende bzw. gewollte Varianz wird ebenfalls erfasst. So wird mit der Erstellung der Produktlinienbeschreibung das *Product Line Scoping* durchgeführt: Es werden die Grenzen der Produktlinie gezogen, d. h., welche der (potenziellen) Produkte durch die Produktlinie unterstützt werden sollen.

Abbildung 3 Produktlinienbeschreibung für CBS.



In Abbildung 3 ist die Produktlinienbeschreibung als Merkmalsmodell (*feature tree*) dargestellt. Dieses Modell ist Ausgangspunkt der nachfolgenden Modellierung der

logischen Architektur und der Softwarearchitekturen für *CBS*. Die hier erfassten Merkmale stellen nur einen beispielhaften Ausschnitt gegenüber der in heutigen Fahrzeugen realisierten *CBS*-Funktionalität dar.

In dem Produktlinienmodell in Abbildung 3 wird festgelegt, dass die Fahrzeugverschleißteile Motoröl, Zündkerzen, Partikelfilter und die vorderen und hinteren Bremsbeläge durch *CBS* beobachtet und bewertet werden sollen. Außerdem soll *CBS* den Termin zur gesetzlichen Fahrzeuguntersuchung (*GFU*) überwachen. Diese Festlegung wird im Merkmalsmodell durch die Merkmale zur `Verschleißermittlung (VE)` repräsentiert. Dabei sind die Umfänge `ZündkerzenVE` und `PartikelfilterVE` abhängig von der technischen Infrastruktur des Fahrzeugs: nur wenn das Fahrzeug einen Otto-Verbrennungsmotor besitzt, sind Zündkerzen im Fahrzeug verbaut, und Partikelfilter gibt es nur bei Dieselmotoren.

Zusätzlich wird festgelegt, welche Beobachtungsmethode für ein einzelnes Verschleißteil angewendet werden soll. Für Motoröl, Partikelfilter und Bremsbeläge wird entsprechende Sensorik zur Bewertung hinzugezogen (Merkmal `Berechnung adaptiv`), bei den Zündkerzen wird zusätzlich die Möglichkeit eingeplant, ein festes Zeit- und Wegintervall (Merkmal `Berechnung linear`) anstatt eines flexiblen Intervalls (`Berechnung adaptiv`) aufgrund von Sensordaten als Bewertungsgrundlage zu verwenden. Diese Situation ist als XOR-Varianz für das Merkmal `ZündkerzenVE` dargestellt.

Die Anzeige des optimalen Servicetermins für den Fahrzeugnutzer erfolgt standardmäßig im Kombiinstrument (Merkmal `KOMBI`). Es wird jedoch auch vorgesehen, das Head-Up-Display (Merkmal `Head-Up Display`) oder das Central Information Display (`CID`) für *CBS* zu verwenden. Da diese nicht in allen Fahrzeugen verbaut sind, sind sie als optional in der Produktlinienbeschreibung gekennzeichnet.

Das *Car Access System* (`CAS`) dient dazu, die ermittelten Verschleiß- und Verfügbarkeitswerte auf den Fahrzeugschlüssel zu kopieren und dem Service so für eine Analyse zur Verfügung zu stellen.

Zum VEIA-Entwicklungsartefakt der Produktlinienbeschreibung gehört auch, die Produkte, die auf Basis der Produktlinie erstellt werden sollen, zu definieren. Diese Informationen werden zum Konfigurieren der weiteren VEIA-Entwicklungsartefakte verwendet.

Folgende drei Produkte werden hier im Folgenden beispielsweise betrachtet:

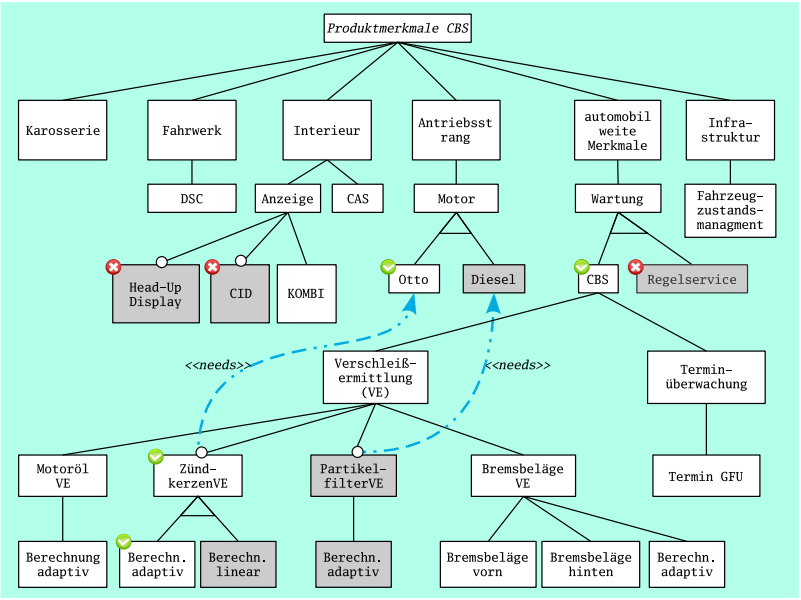
P1: `Ottomotor`, `ZündkerzenVE adaptiv`, keine erweiterte Anzeige (d. h. kein `Head-Up-Display`, kein `CID`) (siehe Abbildung 4),

P2: `Ottomotor`, `ZündkerzenVE linear`, keine erweiterte Anzeige

(d. h. kein `Head-Up-Display`, kein `CID`) (siehe Abbildung 5),

P3: `Dieselmotor`, `PartikelfilterVE adaptiv`, keine erweiterte Anzeige (d. h. kein `Head-Up-Display`, kein `CID`) (siehe Abbildung 6).

Abbildung 4 Produktbeschreibung für Produkt P1.



Anmerkung: Auch wenn die erweiterten Anzeigemöglichkeiten (Merkmale `CID` und `Head-Up Display`) derzeit nicht in einem der oben angegebenen Produkte verwendet sind, werden sie dennoch als Varianten in den Architekturmodellen bereits berücksichtigt.

Abbildung 5 Produktbeschreibung für Produkt P2.

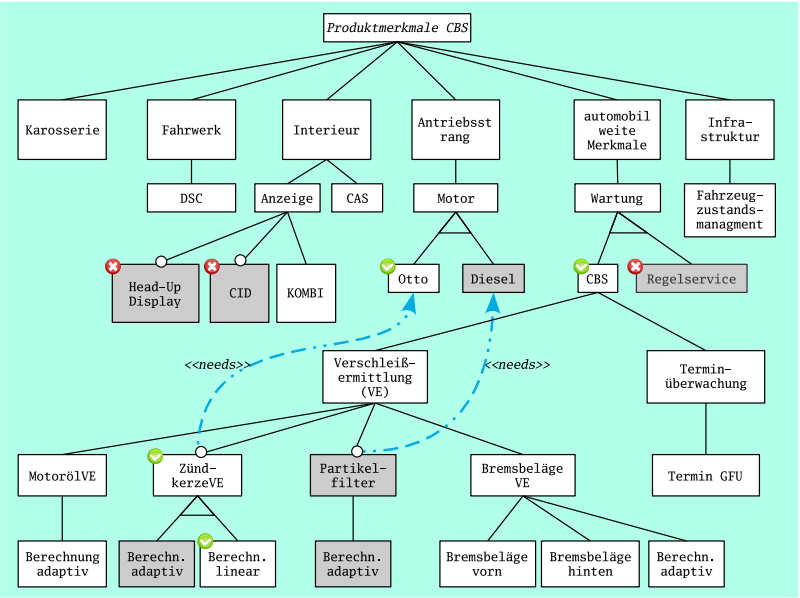
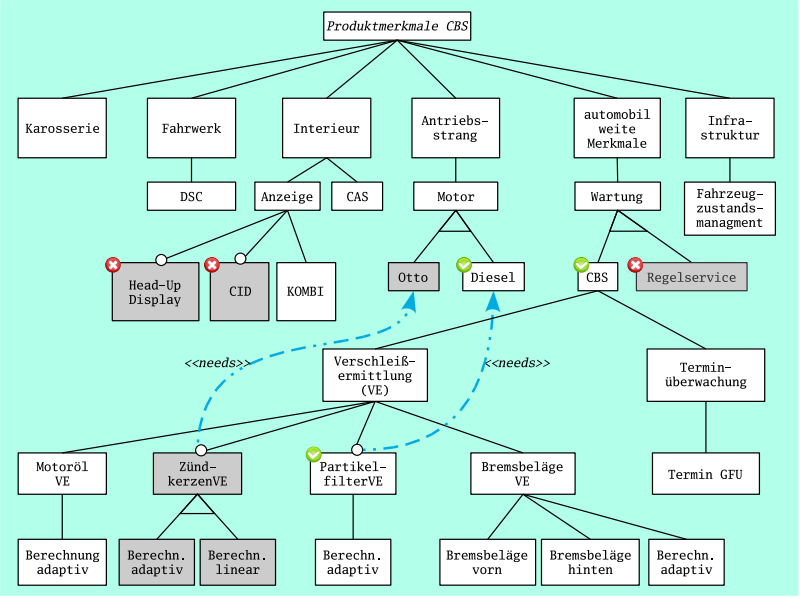


Abbildung 6 Produktbeschreibung für Produkt P3.



4 Logische Architektur

Im Folgenden erfolgt die funktionale Beschreibung für CBS. Die logische Architektur berücksichtigt alle durch die Produktlinienbeschreibung vorgegebenen CBS-Umfänge.

4.1 Funktionsnetz CBS

Das Funktionsnetz für das »System« CBS wird in Abbildung 7 beschrieben: mit den notwendigen Schnittstellen zur Systemumgebung, d. h. mit der Festlegung des Informationsbedarfs und des Informationsangebots der Funktion `CBS`. In dieser Abbildung ist `CBS` als Black-Box dargestellt.

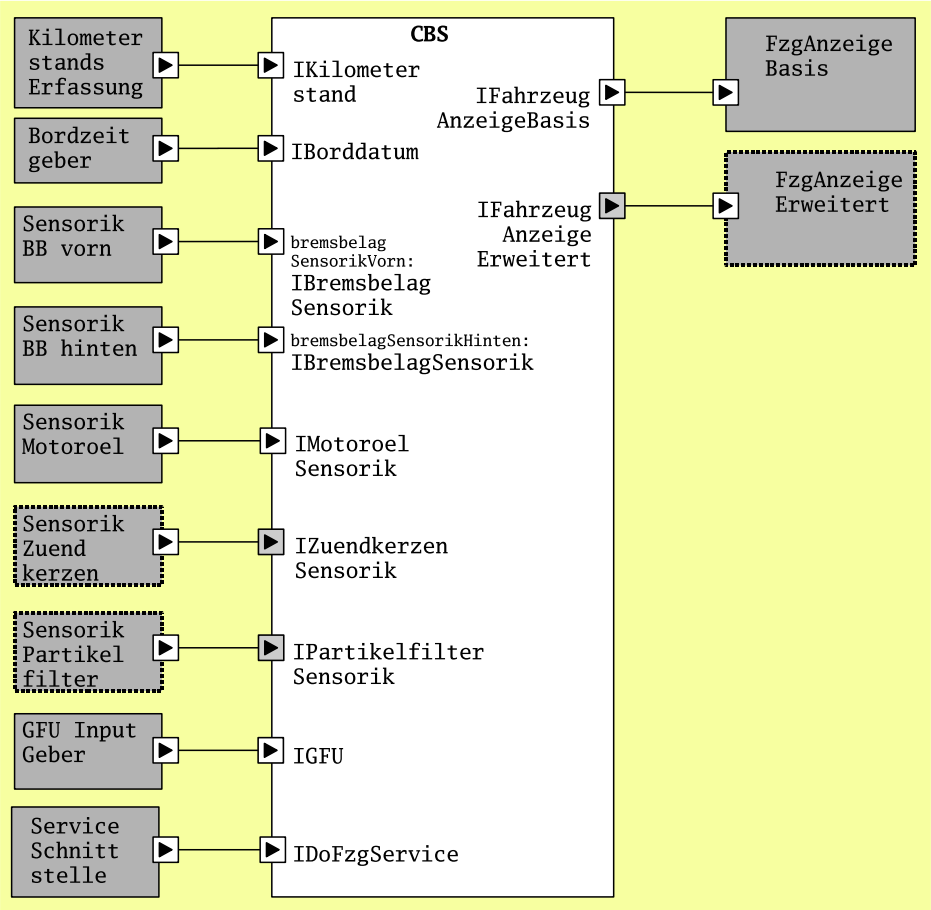
In Abbildung 8 wird `CBS` in Unterfunktionen verfeinert.[3] Ein Teil der Unterfunktionen repräsentiert dabei jeweils einen der geplanten CBS-Umfänge. Diese Unterfunktionen beschreiben funktional die Verschleißerfassung und -überwachung des jeweiligen Verschleißteils.

Wie bereits in der Produktlinienbeschreibung festgelegt ist, werden die Überwachung der CBS-Umfänge für die vorderen und hinteren Bremsbeläge fest in der Produktlinienarchitektur eingeplant. Dies gilt auch für die Überwachung der Qualität des Motoröls und des Einhaltens des Termins der gesetzlichen Fahrzeuguntersuchung. Partikelfilter können nur überwacht werden, wenn das Fahrzeug über einen Dieselmotor verfügt, deshalb ist die Verschleißüberwachung als optional gekennzeichnet. Analoges gilt für die Zündkerzenüberwachung, sie kann nur bei Vorhandensein von Ottomotoren erfolgen.

Dass Otto- und Dieselmotoren sich gegenseitig ausschließen, könnte in der logischen Architektur mit Hilfe eines Varianzpunkts dargestellt werden, der die Überwachung von Partikelfilter und Zündkerzen alternativ zueinander setzt. Je-

[3] Die Delegationsverbindungen zwischen der Funktion `CBS` und ihrer Unterfunktionen, die jeweils die Ports zum Kilometerstand und zum Borddatum miteinander verbinden, sind in der Abbildung aus Übersichtlichkeitsgründen nur angedeutet. Ebenso verhält es sich mit den Konnektoren zwischen der Funktion `CBS Serviceintervall Rücksetzen` und den Funktionen für die CBS-Umfänge bzgl. der Ports mit der Portdeklaration `IDoService`.

Abbildung 7 Funktion CBS (Schnittstellen und Umgebung).

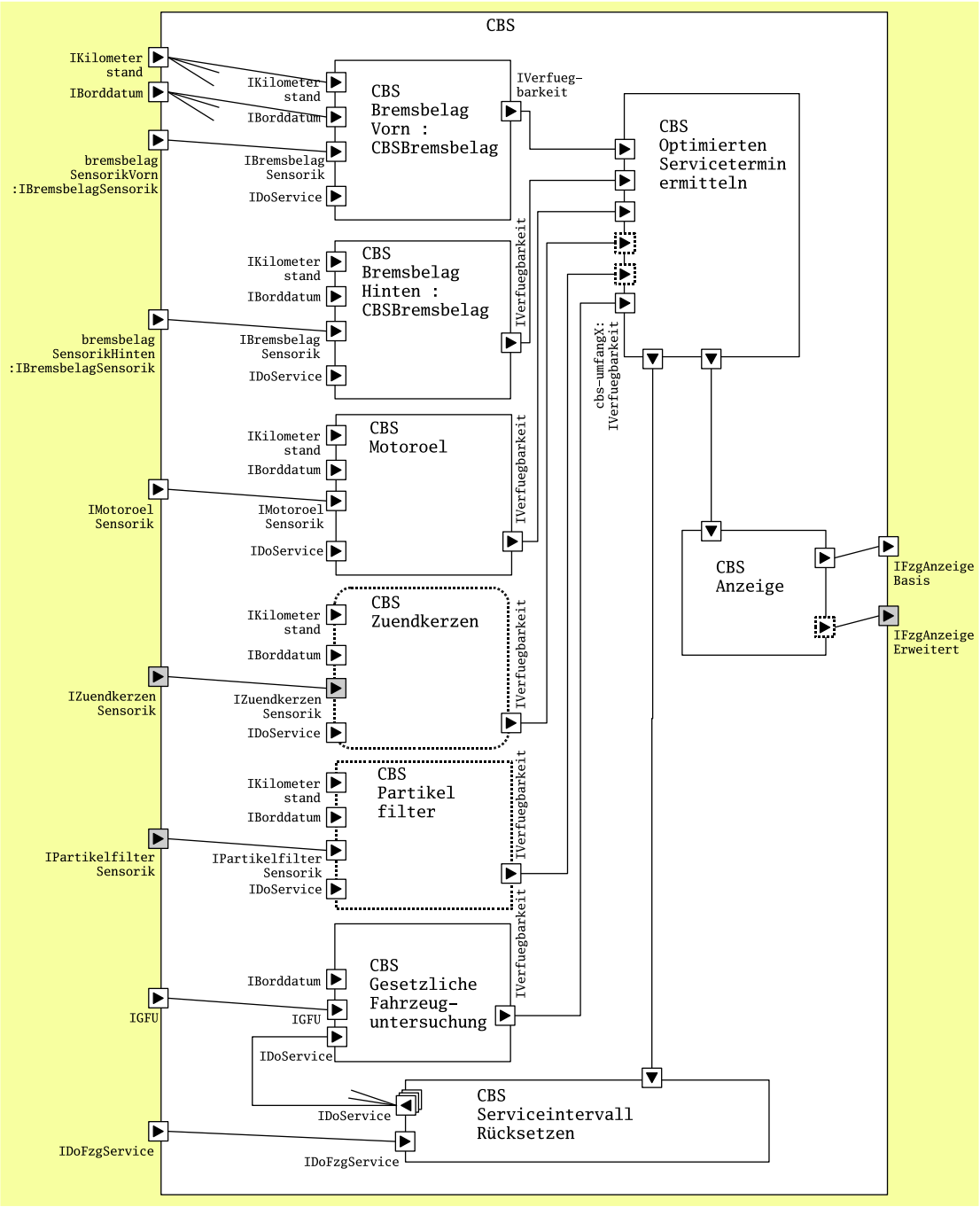


doch stellt diese Situation keine funktionale Varianz dar, sondern hängt von der Varianz der Infrastruktur (der technischen Architektur) ab. Deshalb wird funktional von einer unabhängigen Varianz ausgegangen, hier als Option dargestellt. Durch die Verteilung und Konfiguration werden dann die jeweils benötigten Funktionen ausgewählt und die nicht notwendigen Funktionen abgewählt.[4]

Zusätzlich zur Optionalität der Zündkerzenüberwachung wird die Funktion außerdem als Varianzpunkt spezifiziert: ihre Varianten entsprechen den in der Produkt-

[4] In der graphischen Notation sind Optionen durch gestrichelte Linien gekennzeichnet.

Abbildung 8 Funktionsnetz CBS-Umfänge.



linienbeschreibung festgelegten Anforderungen, dass Zündkerzen alternativ mit festen Weg- bzw. Zeitintervallen überprüft werden sollen (lineare Zündkerzenüberwachung) oder auf Basis entsprechender Sensorik verschleißabhängig überwacht werden (adaptive Zündkerzenverschleißerfassung). Auf funktionaler Ebene wird spezifiziert, was der Informationsbedarf der jeweiligen Variante ist.[5]

Ein anderer Teil der CBS-Unterfunktionen (die Unterfunktionen `CBS Optimierten Servicetermin ermitteln` und `CBS Anzeige`) beschreibt, wie die Verschleißwerte der einzelnen Umfänge zu einem einzigen vorgeschlagenen optimalen Servicetermin zusammengefasst und dann im Fahrzeug angezeigt werden.

An dieser Stelle ist anzumerken, dass noch nicht vorweg genommen wurde, *wie* die Funktionalität *CBS* durch Software zu implementieren ist, d. h., wie sie in entsprechende Softwarekomponenten aufzuteilen ist. Es ist rein funktional beschrieben, dass einzelne Verschleißteile zu überwachen sind und die Werte dann zusammengefasst werden müssen.

4.2 CBS-Umfang Bremsbelag

Im Folgenden wird die CBS-Unterfunktion `CBSBremsbelag`, die in der logischen Architektur zweimal – für die vorderen und für die hinteren Bremsbeläge – verwendet wird, funktional verfeinert.

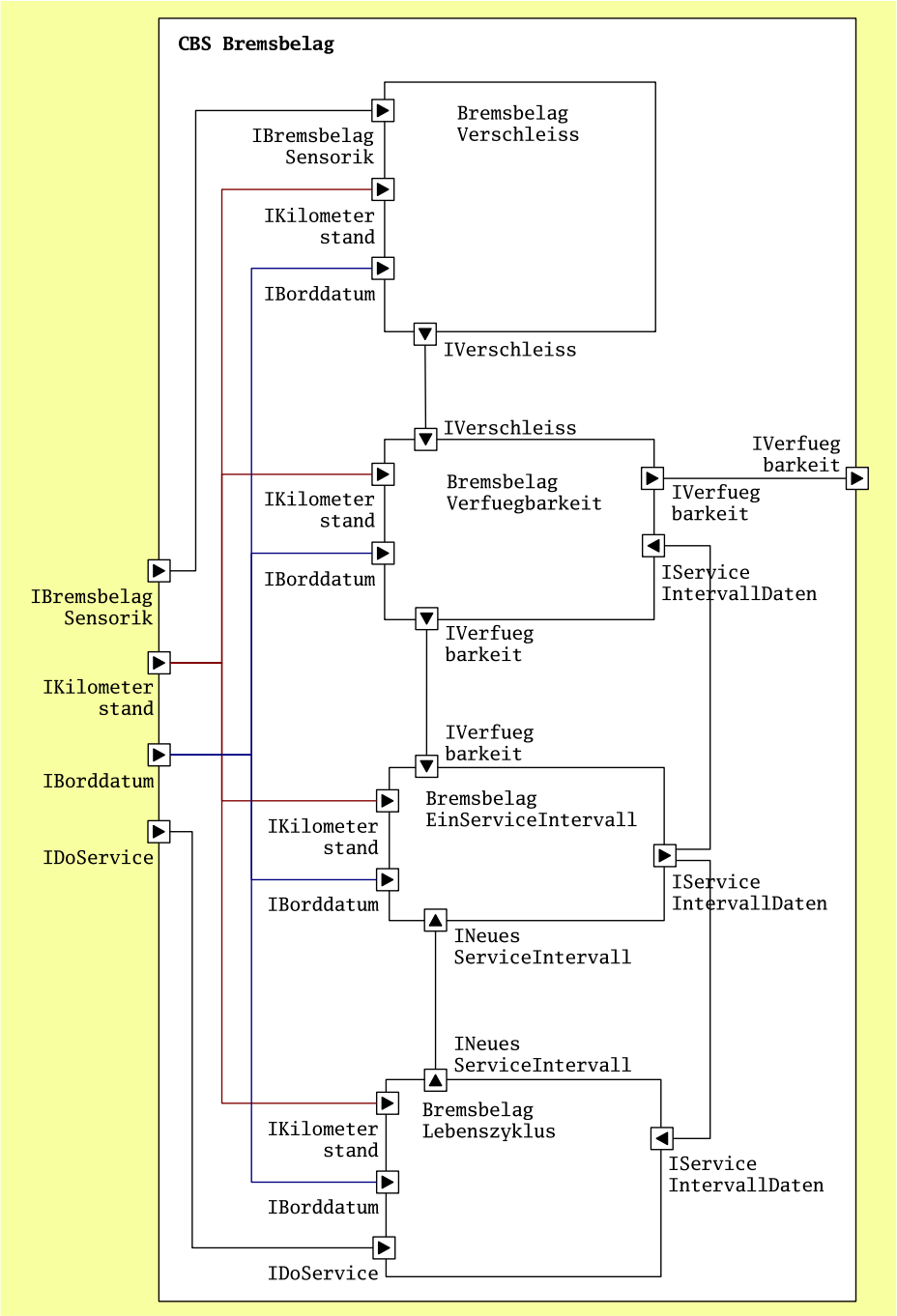
4.2.1 Funktionsnetz `CBSBremsbelag` (vorn oder hinten)

In Abbildung 9 sind die Funktionen für die Prognose der Verfügbarkeit für Bremsbeläge dargestellt, die als hierarchische Funktion `CBSBremsbelag` zusammengefasst sind. Die Funktion `CBSBremsbelag` ist in folgende Unterfunktionen aufgeteilt:

`BremsbelagVerschleiss` bestimmt aus den Sensorwerten der gegebenen Sensorik den aktuellen Verschleiß.

[5] In der graphischen Notation sind Funktionsvarianzpunkte durch abgerundete Rechtecke gekennzeichnet.

Abbildung 9 Funktionsnetz CBS-Umfang Bremsbelag (vorn oder hinten)



`BremsbelagVerfuegbarkeit` bestimmt aus dem aktuellen Verschleiß die prozentuale Verfügbarkeit.

`BremsbelagEinServiceIntervall` bestimmt innerhalb eines laufenden Serviceintervalls auf Basis der übermittelten Verfügbarkeit die Weg- und/oder Zeitprognose. Ggf. wird die Prognose auf den aktuellen Verschleiß angepasst.

`BremsbelagLebenszyklus` verwaltet (historisiert) die Serviceintervalle mit den erfassten Daten zu gefahrerener Wegstrecke, Dauer des Serviceintervalls, erfasstem Verbrauch und prognostizierter Verfügbarkeit.

4.2.2 Portdeklarationen

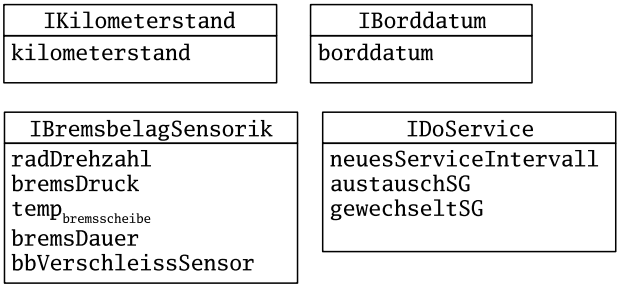
Portdeklarationen für externe Ports

Die Portdeklarationen für die Ports der Funktion `CBSBremsbelag` ist in Abbildung 10 dargestellt.

Um den Verschleiß zu prognostizieren, benötigt die Funktion Informationen

- zur vom Fahrzeug zurückgelegten Wegstrecke (`IKilometerstand`),
- zur Bordzeit (um die Wegstrecke mit der dafür benötigten Zeit in Beziehung zu setzen) (`IBorddatum`),
- von der vorhandenen Sensorik der Bremsbeläge (`IBremsbelagSensorik`) und
- zu Serviceanforderungen (z. B. Start eines Serviceintervalls, Durchführung von Wartungsarbeiten etc.) (`IDoService`).

Abbildung 10 CBS-Portdeklarationen (Bremsbelag extern)

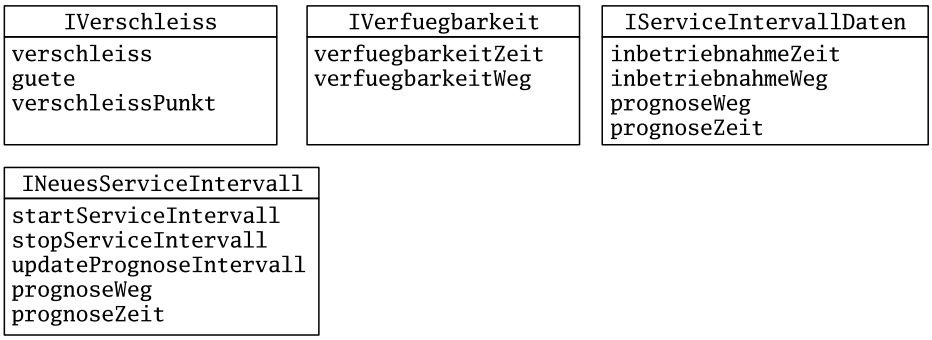


Portdeklarationen für interne Ports

Die Funktion `CBSBremsbelag` wird funktional in weitere Unterfunktionen verfeinert, die miteinander kooperieren und demzufolge kommunizieren. Die Portdeklarationen für die entsprechenden Ports seien an dieser Stelle angegeben, siehe Abbildung 11:

- ermittelter Verschleiß der Bremsbeläge (`IVerschleiss`),
- prognostizierte Verfügbartkeit der Bremsbeläge (in Form von Zeit- und/oder Wegangaben) (`IVerfuegbarkeit`),
- Informationen zum gültigen Serviceintervall (Start des Intervalls, angenommener Verschleiß- bzw. Verfügbarkeitsverlauf) (`IServiceIntervallDaten`) und
- Informationen zum Start eines neues Serviceintervalls (`INeuesServiceIntervall`).

Abbildung 11 CBS-Portdeklarationen (Bremsbelag intern)



4.3 CBS-Umfang Motoröl

Motoröl ist ein weiterer Umfang, der von `CBS` beobachtet werden soll. Die Funktion `CBSMotoroel` erfüllt eine ähnliche Funktion wie `CBSBremsbelag`, jedoch bewertet sie den Verschleiß für das Motoröl.

4.3.1 Funktionsnetz `CBSMotoroel`

Das Funktionsnetz ist in Abbildung 12 dargestellt.

4.3.2 Portdeklarationen

Von der Motorölsensorik werden Informationen zum Kraftstoffverbrauch und zum Ölwechselzeitpunkt benötigt (`IMotoroelSensorik`).

Außerdem werden die schon beschriebenen Informationen zu Kilometerstand, Bordzeit und Serviceanforderungen benötigt (vgl. Abschnitt 4.2).

4.4 CBS-Umfang Zündkerzen

An dieser Stelle die funktionale Beschreibung für die Verschleißermittlung der Zündkerzen.

4.4.1 Funktionsnetz `CBSZündkerzen`

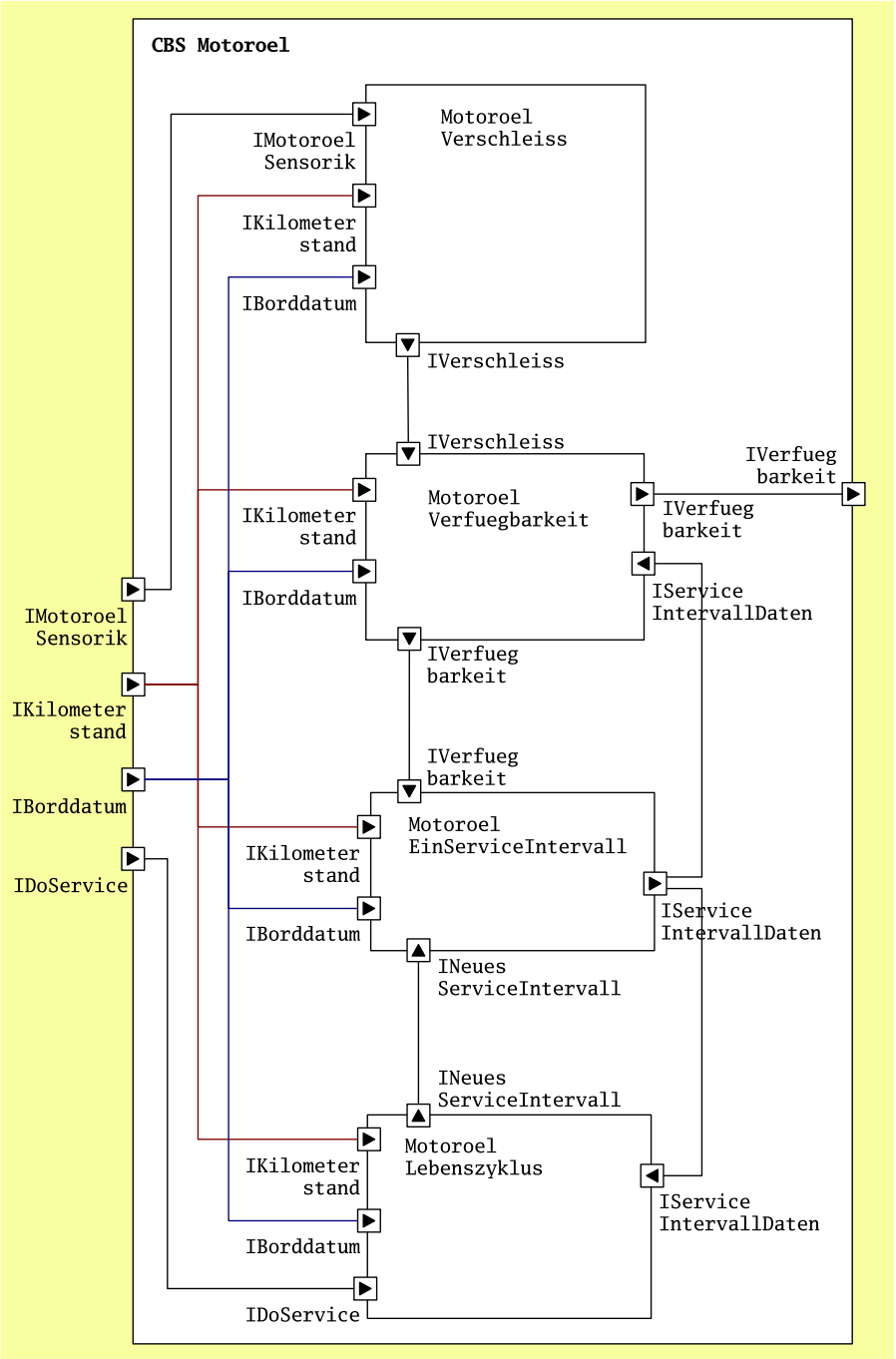
Die Funktion `CBSZündkerzen` ist in der Produktlinie als optional gekennzeichnet, weil sie nur bei Vorhandensein von Ottomotoren sinnvoll ist und nur dann verbaut bzw. ausgeführt werden soll.

Des weiteren wird die Verschleißüberwachung für Zündkerzen in zwei Varianten geplant: mit festen Serviceintervallen (lineare Variante) bzw. mit variablen Serviceintervallen (adaptive Variante) unter Berücksichtigung von Informationen der Zündkerzensensorik. Vgl. hierzu die Produktlinienbeschreibung in Kapitel 3.

Die funktionale Beschreibung hierzu sieht wie folgt aus:

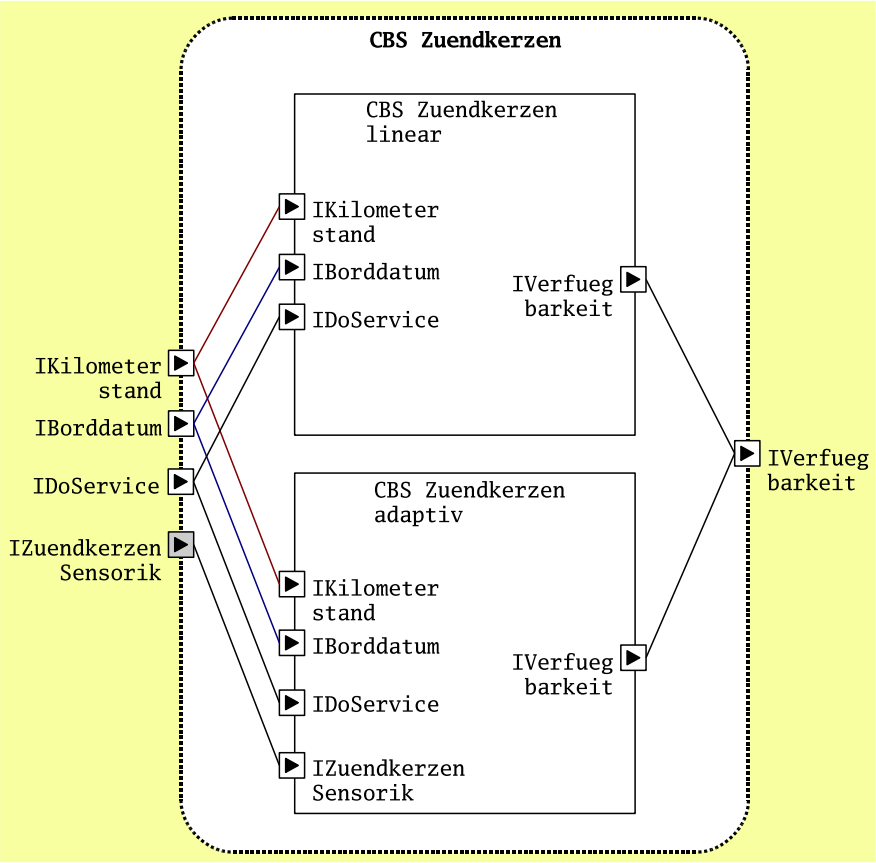
- der Funktionsvarianzpunkt `CBSZuendkerzen` mit den zwei Alternativen `CBSZuendkerzenlinear` und `CBSZuendkerzenadaptiv` siehe Abbildung 13,
- die Verfeinerung der linearen Variante `CBSZuendkerzenlinear` siehe Abbildung 14,

Abbildung 12 Funktionsnetz CBS-Umfang Motoröl



– die Verfeinerung der adaptiven Variante `CBSZuendkerzenadaptiv` siehe Abbildung 15.

Abbildung 13 Funktionsnetz CBS-Umfang Zündkerzen (Varianzpunkt).



4.5 CBS-Umfang Gesetzliche Fahrzeuguntersuchung (GFU)

Vom Gesetzgeber sind regelmäßige, in festen Zeitintervallen durchzuführende Fahrzeuguntersuchungen vorgeschrieben. Solche Termine sind die Hauptuntersuchungen sowie die Abgasuntersuchungen. Die Funktion `CBS` soll auch die Einhaltung solcher gesetzlich vorgeschriebenen Termine überwachen und mit Terminen für durch Verschleiß bedingte Werkstättenbesuche abstimmen.

Zur Terminkontrolle der gesetzlichen Untersuchungen wird hier beispielhaft die Funktion `CBSGFU` spezifiziert. Im Gegensatz zu Verschleißteilen wird hier keine

Abbildung 14 Funktionsnetz CBS-Umfang Zündkerzen (lineare Variante).

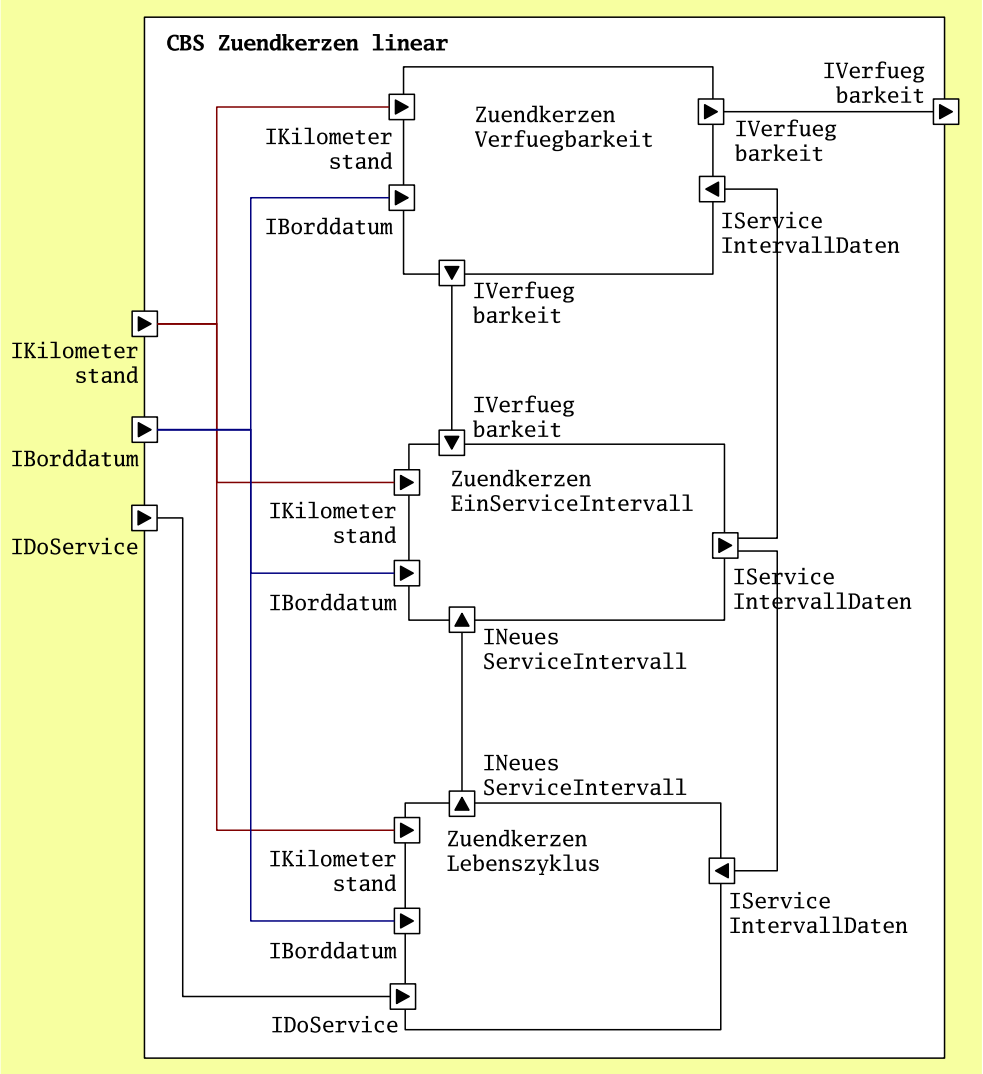
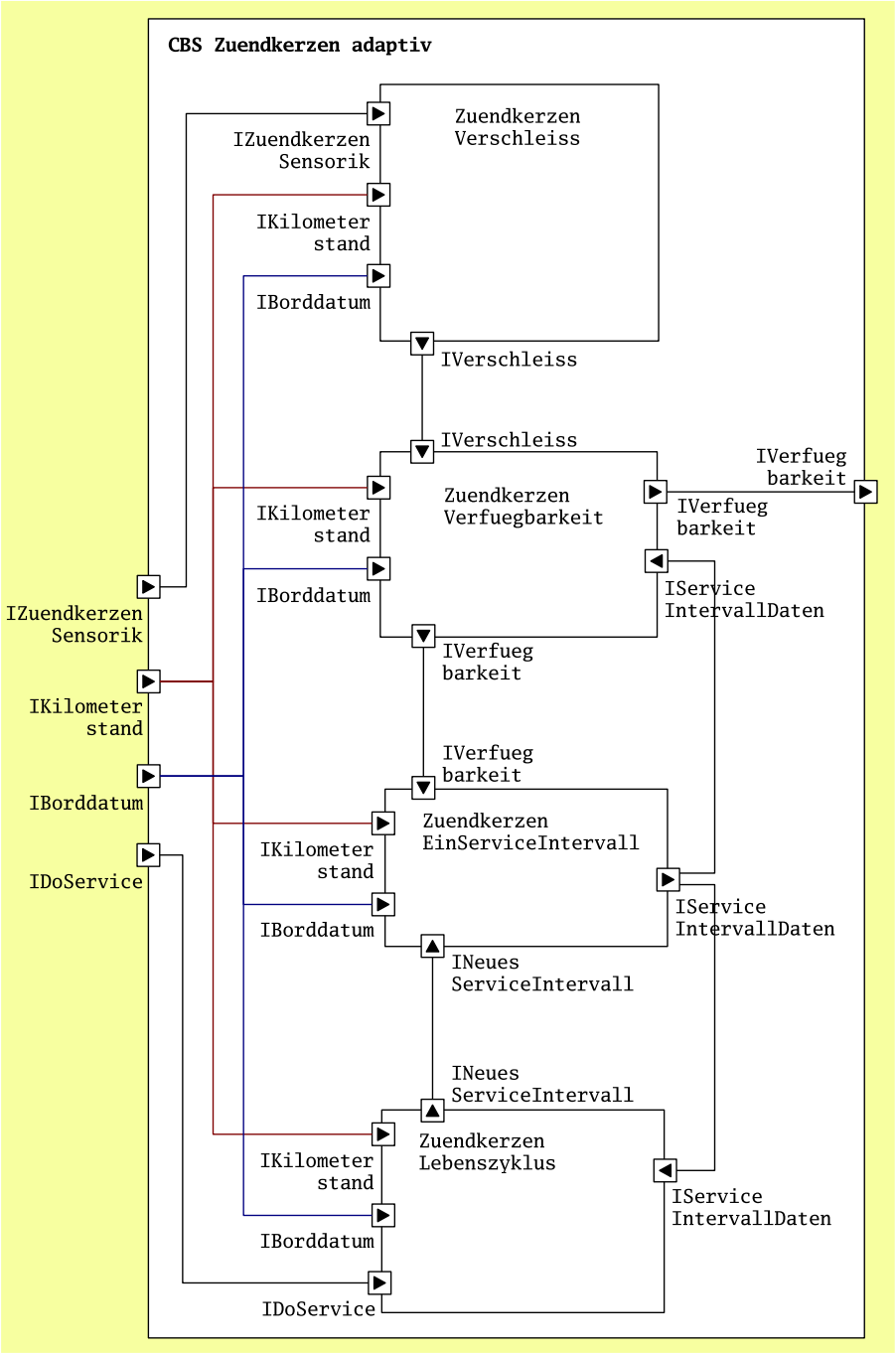


Abbildung 15 Funktionsnetz CBS-Umfang Zündkerzen (adaptive Variante).

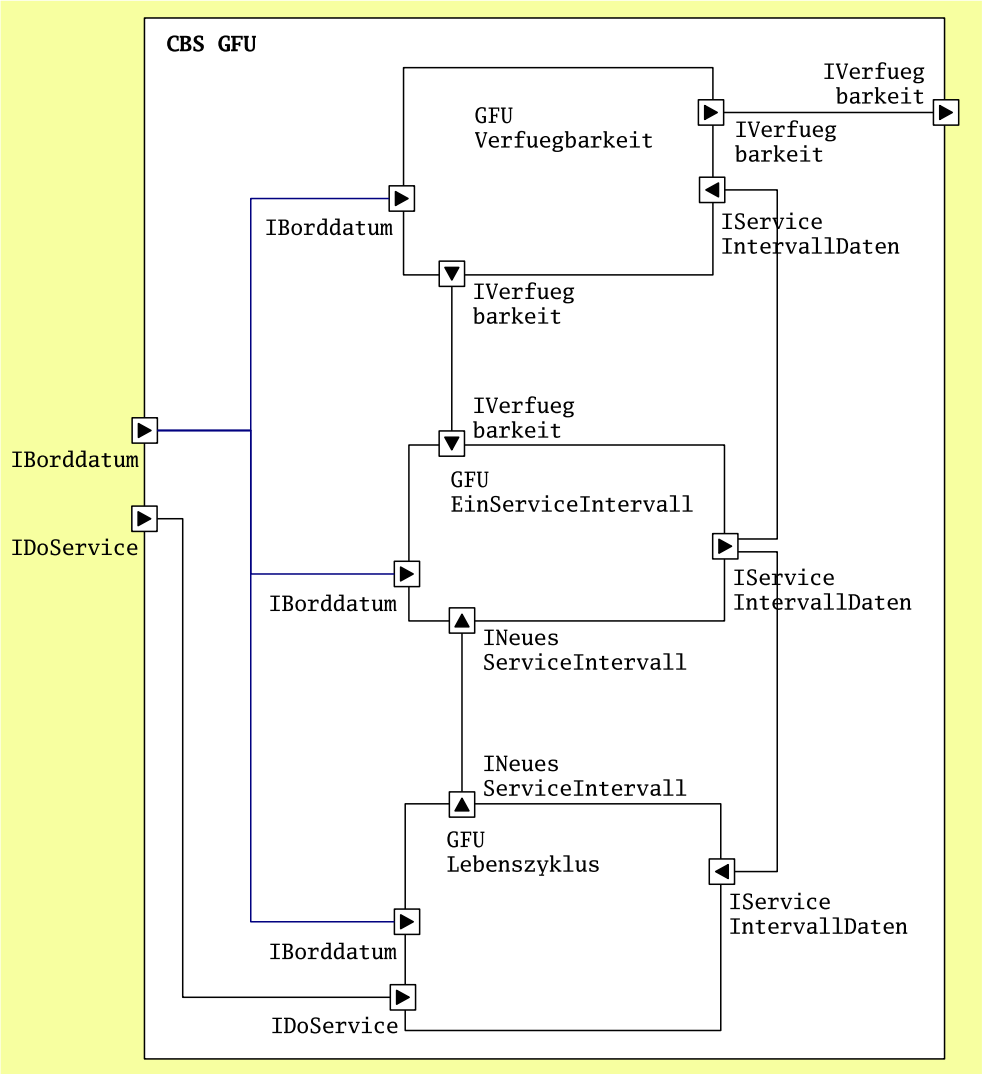


weitere Verschleißsensorik benötigt, sondern nur das Borddatum ausgewertet.

4.5.1 Funktionsnetz CBSGFU

Das Funktionsnetz für den Umfang der gesetzlichen Fahrzeuguntersuchung ist in Abbildung 16 dargestellt.

Abbildung 16 Funktionsnetz CBS-Umfang für die gesetzliche Fahrzeuguntersuchung (GFU).



5 Softwarearchitekturen

Softwarearchitekturen als VEIA-Entwicklungsartefakte setzen die in der logischen Architektur festgelegte Funktionalität in Software um, indem entsprechende Softwarekomponenten spezifiziert werden. Die strukturelle Aufteilung kann dabei unterschiedlich zu der in der logischen Architektur sein, weil in der Softwarearchitektur weitere Aspekte eine Bedeutung spielen können.

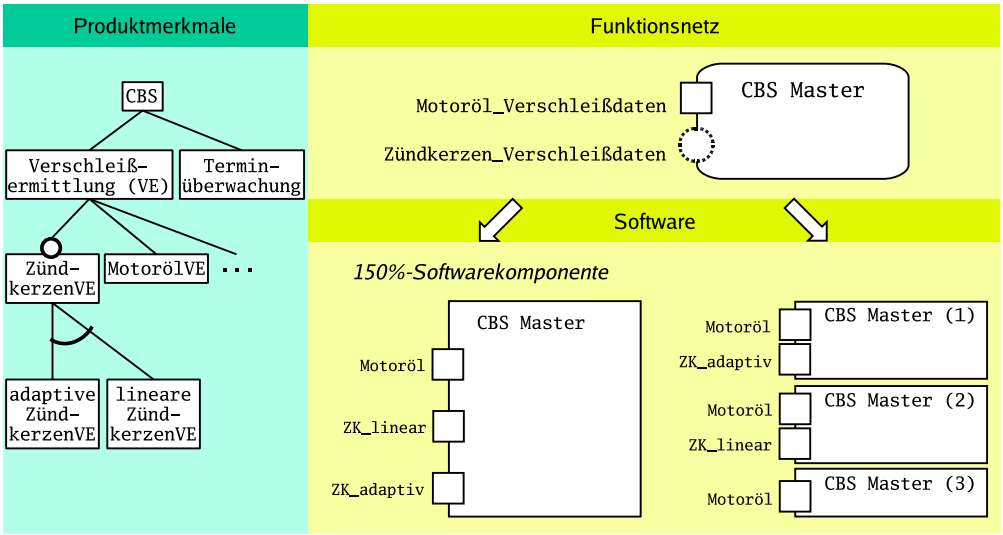
Die in der logischen Architektur definierte Varianz muss durch Software geeignet realisiert werden. Hierzu gehört, dass entsprechend geplant wird,

- wann die Varianz gebunden wird (*Binding Time* von Varianz),
- mit welcher Technologie die Varianz implementiert wird.

Dies hat Einfluss auf die Ausgestaltung einer Softwarearchitektur. Im Rahmen der Fallstudie zu *CBS* gehen wir von zwei prinzipiellen Realisierungsmöglichkeiten eines Produktlinienfunktionsnetzes aus:

1. produktgenerische Softwarearchitekturen und -komponenten vs.
2. produktspezifische Softwarearchitekturen und -komponenten

Abbildung 17 Vom Funktionsnetz zur Softwarearchitektur.



Beide Möglichkeiten stellen die Extrempunkte dar, im Allgemeinen werden Lö-

sungen dazwischen zum Zuge kommen. Beide Realisierungsmöglichkeiten sind in Abbildung 17 illustriert.

Im Folgenden werden die beiden Möglichkeiten für die Ausgestaltung der Softwarearchitektur für das zuvor definierte Funktionsnetz ausgearbeitet.

5.1 Produktgenerische Softwarearchitektur und Softwarekomponenten

Die erste Möglichkeit, die zu unterstützende Varianz und damit alle Produkte einer Produktlinie durch die Software zu realisieren, ist die generische Gestaltung der Software. Dies geschieht so, dass sämtliche Variationen durch Software umgesetzt werden und ggf. durch (Konfigurations-) Parameter angepasst werden. Bei BMW wird in diesem Zusammenhang häufig von einem 150 %-Vorgehen gesprochen.

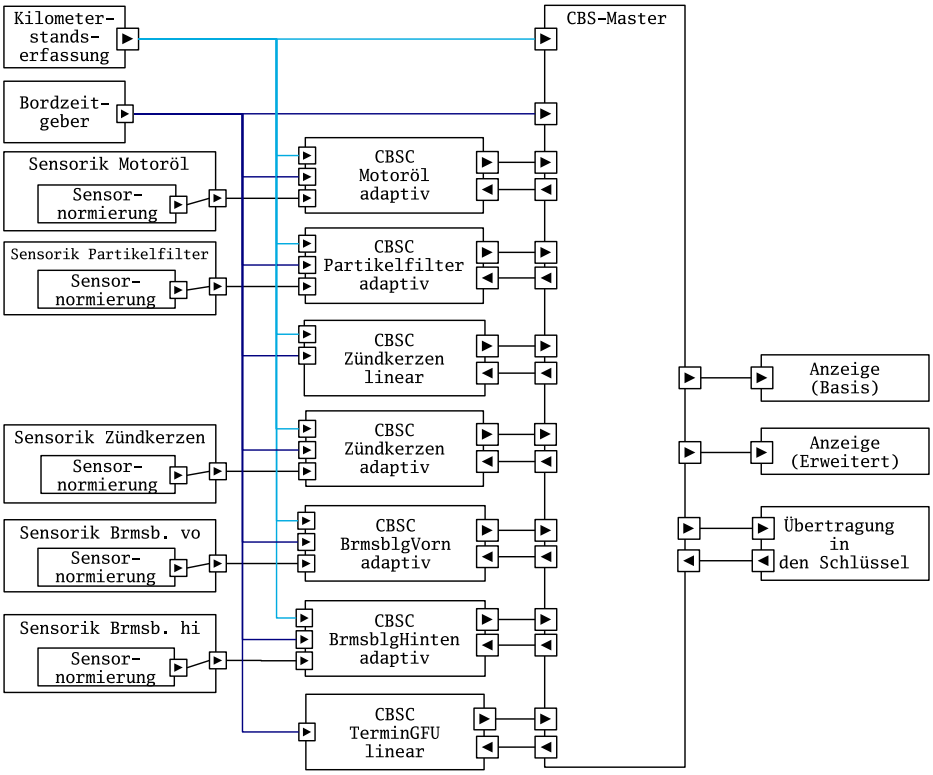
Ein Vorteil dieses Ansatzes ist sicherlich: *"One fits all."* Die Architektur und die Komponenten werden einmal entwickelt, sie passen für jedes Produkt. Ein möglicher Nachteil ist, dass in einem Produkt zu viel an Funktionalität verbaut wird (Stichwort »toter Code«), die jedoch nie im Produktlebenszyklus ausgeführt werden soll.

In Abbildung 18 ist die Softwarearchitektur hierzu dargestellt: Sämtliche Varianz des Funktionsnetzes ist durch Ausmultiplizieren »aufgelöst«, zur Ausführungszeit der Software (nicht dargestellt) werden jedoch einige Komponenten deaktiviert sein. Die Softwarekomponenten sind hier nicht mehr direkt durch Konfiguration aus- oder abwählbar. Allenfalls durch Verteilung der Softwarekomponenten auf die Varianz enthaltene technische Architektur können Softwarekomponenten nicht in ein Produkt verbaut werden, weil das entsprechende Steuergerät auch nicht im Produkt verbaut ist. Beispiel hierzu sind die Softwarekomponenten für die Überwachung der Zündkerzen als auch für den Partikelfilter, weil entweder nur das `DME` oder nur das `DDE` in einem Produkt verbaut wird.

Abbildung 19 zeigt die Verfeinerung der Softwarekomponente `CBSMaster`, die andeutet, dass beispielsweise auch mehr Datenspeicher eingeplant werden muss, als in einem Produkt je benötigt werden wird.

Allein daraus, dass nicht jedes Steuergerät der Produktlinie in ein Fahrzeug verbaut wird, fallen diejenigen Softwarekomponenten aus der Softwarearchitektur heraus, die auf solche Steuergeräte alloziert werden. In Abbildung 20 ist musterhaft dargestellt, dass alle adaptiven CBS-Clients auf dem zugehörigen Steuergerät und alle linearen CBS-Clients auf dem Kombiinstrument installiert werden. Dadurch, dass beispielsweise nicht immer das `DDE` verbaut ist, ist demzufolge die Softwarekomponente `CBSCPartikelfilteradaptiv` auch nicht immer in einem Produkt

Abbildung 18 Softwarearchitektur: CBS-150 %-Architektur (gesamt).



vorhanden.

Dies ist durch die folgenden Softwarearchitekturmodelle für die Produkte 1 bis 3 illustriert. Dabei ist die Softwarekomponente `CBSMaster` für alle drei Produkte identisch, vgl. Abbildung 19, ggf. wird die Komponente produktspezifisch über Parameter konfiguriert.

5.1.1 Softwarearchitektur für Produkt 1

Gemäß der Produktbeschreibung in Kapitel 3, Abbildung 4 ist die in Abbildung 21 abgebildete Architektur für das Produkt 1 gültig.

5.1.2 Softwarearchitektur für Produkt 2

Gemäß der Produktbeschreibung in Kapitel 3, Abbildung 5 ist die in Abbildung 22 abgebildete Architektur für das Produkt 2 gültig.

5.1.3 Softwarearchitektur für Produkt 3

Gemäß der Produktbeschreibung in Kapitel 3, Abbildung 6 ist die in Abbildung 23 abgebildete Architektur für das Produkt 3 gültig.

5.2 Produktspezifische Softwarearchitekturen und Softwarekomponenten

Die zweite hier betrachtete Möglichkeit, die zu unterstützende Varianz und damit alle Produkte einer Produktlinie durch die Software zu realisieren, ist, die produktspezifische Software mit Hilfe eines Softwareproduktlinienansatzes zu erzeugen. Auf Basis einer Produktlinienarchitektur und entsprechenden Softwarekomponenten(-modellen) wird für jedes Produkt die zu ihr passende Software erstellt. Die Software in einem Produkt enthält keine Varianz mehr, sie ist bei Erzeugung der Software aufgelöst wurden, indem die entsprechende Variante gewählt wurde.

Vorteil ist, dass kein »toter Code« verbaut wird, so dass die Softwarekomponenten schlank und im Allgemeinen kleiner sind als beim produktgenerischen Ansatz (was sich positiv auf die Flashzeiten auswirken kann). Möglicher Nachteil ist jedoch, dass das Erzeugen der Softwarekomponenten technologisch aufwändiger ist (man benötigt unter Umständen Generatoren), dass – logistisch gesehen – mit einer größeren Anzahl an Softwarekomponenten umgegangen werden muss (für jedes Produkt gibt es unter Umständen spezifische Komponenten), und dass ein Produkt unter Umständen schwerer nachzurüsten ist.

5.2.1 Softwareproduktlinienarchitektur

In Abbildung 24 ist die Softwareproduktlinienarchitektur dargestellt, in Abbildung 25 ist die noch Varianz enthaltene Softwarekomponente `CBSMaster` abgebildet.

Im Folgenden wird die jeweils gültige Architektur für die drei geplanten Produkte, die aus der Softwareproduktlinienarchitektur erzeugt wird, vorgestellt.

5.2.2 Softwarearchitektur und Softwarekomponenten für Produkt 1

Gemäß der Produktbeschreibung in Kapitel 3, Abbildung 4 wird aus der Produktlinienarchitektur, vgl. Abbildung 24 und Abbildung 25, die in Abbildung 26 und Abbildung 27 abgebildete Architektur mit den zugehörigen Softwarekomponenten erzeugt.

5.2.3 Softwarearchitektur und Softwarekomponenten für Produkt 2

Gemäß der Produktbeschreibung in Kapitel 3, Abbildung 5 wird aus der Produktlinienarchitektur, vgl. Abbildung 24 und Abbildung 25, die in Abbildung 28 und Abbildung 29 abgebildete Architektur mit den zugehörigen Softwarekomponenten erzeugt.

5.2.4 Softwarearchitektur und Softwarekomponenten für Produkt 3

Gemäß der Produktbeschreibung in Kapitel 3, Abbildung 6 wird aus der Produktlinienarchitektur, vgl. Abbildung 24 und Abbildung 25, die in Abbildung 30 und Abbildung 31 abgebildete Architektur mit den zugehörigen Softwarekomponenten erzeugt.

Abbildung 19 Softwarearchitektur: CBS-150 %-Architektur (Master).

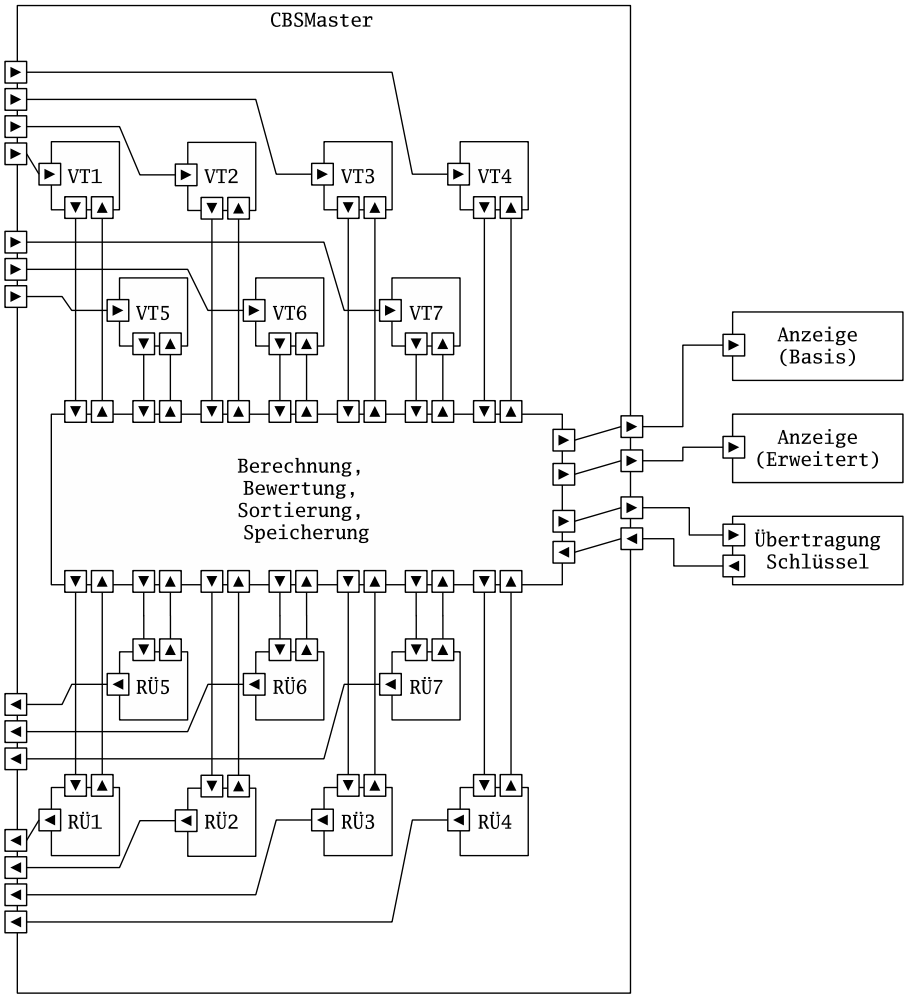


Abbildung 20 Allokation der Softwarekomponenten aus die Hardwarearchitektur.

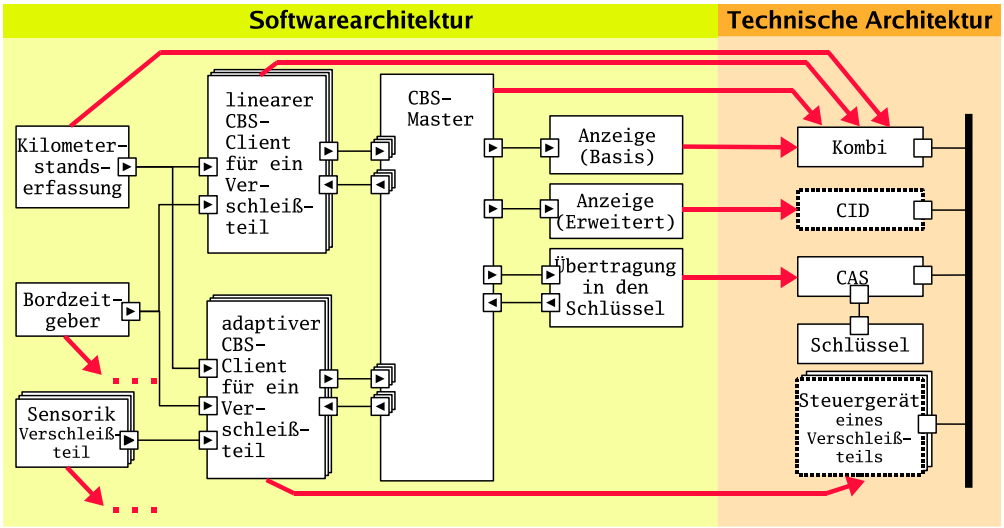


Abbildung 21 Softwarearchitektur: Produkt 1 auf Basis des produktgenerischen Ansatzes (gesamt).

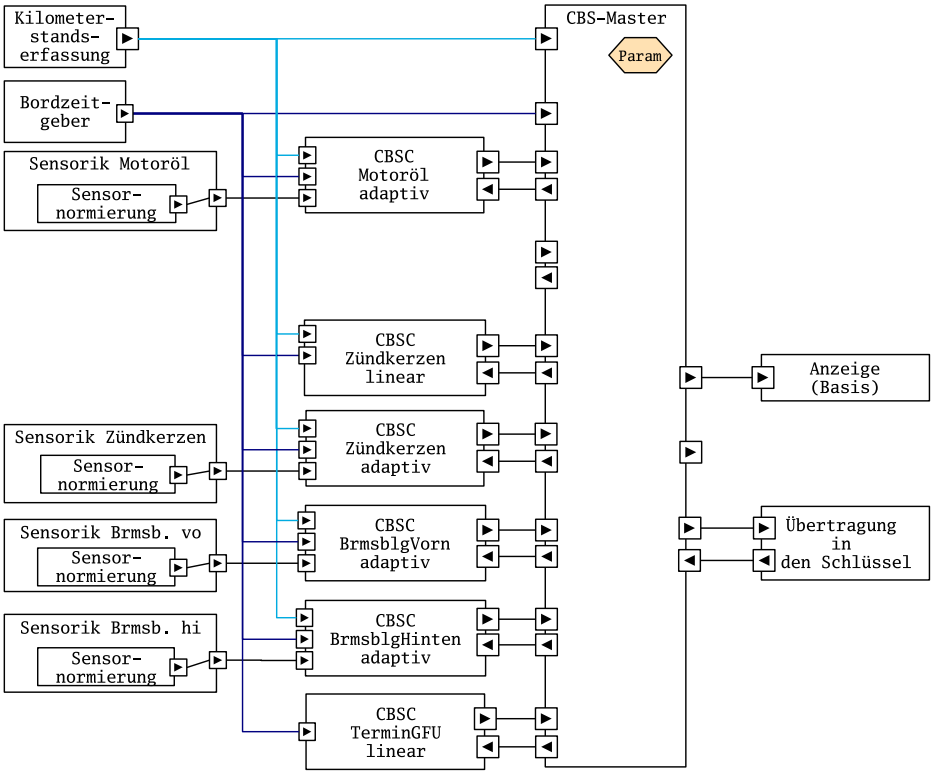


Abbildung 22 Softwarearchitektur: Produkt 2 auf Basis des produktgenerischen Ansatzes (gesamt).

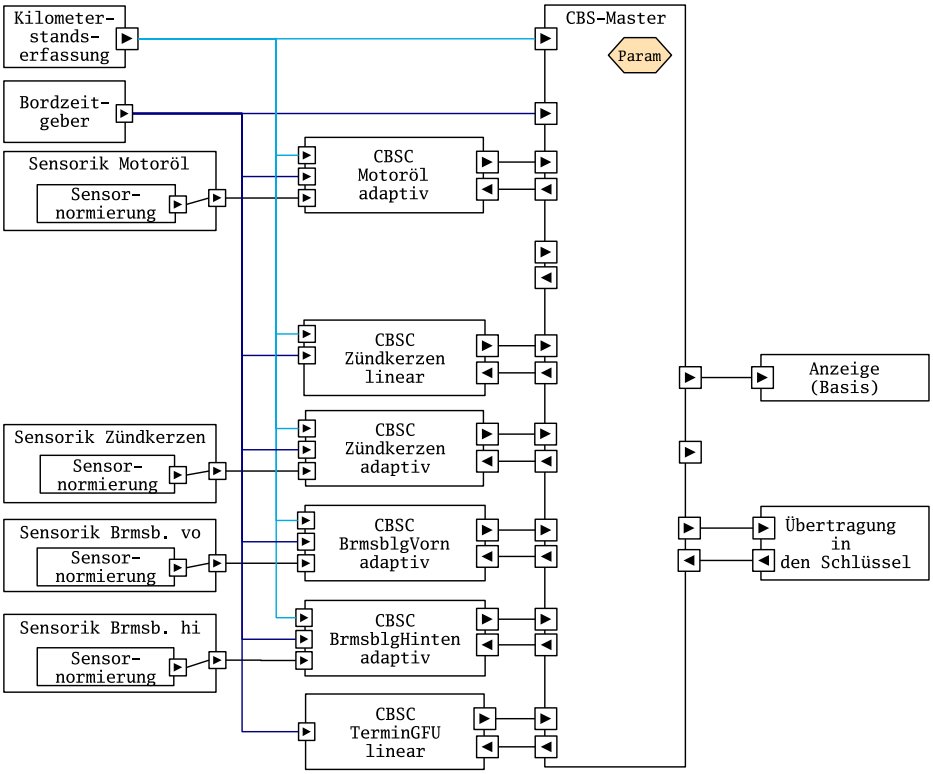


Abbildung 23 Softwarearchitektur: Produkt 3 auf Basis des produktgenerischen Ansatzes (gesamt).

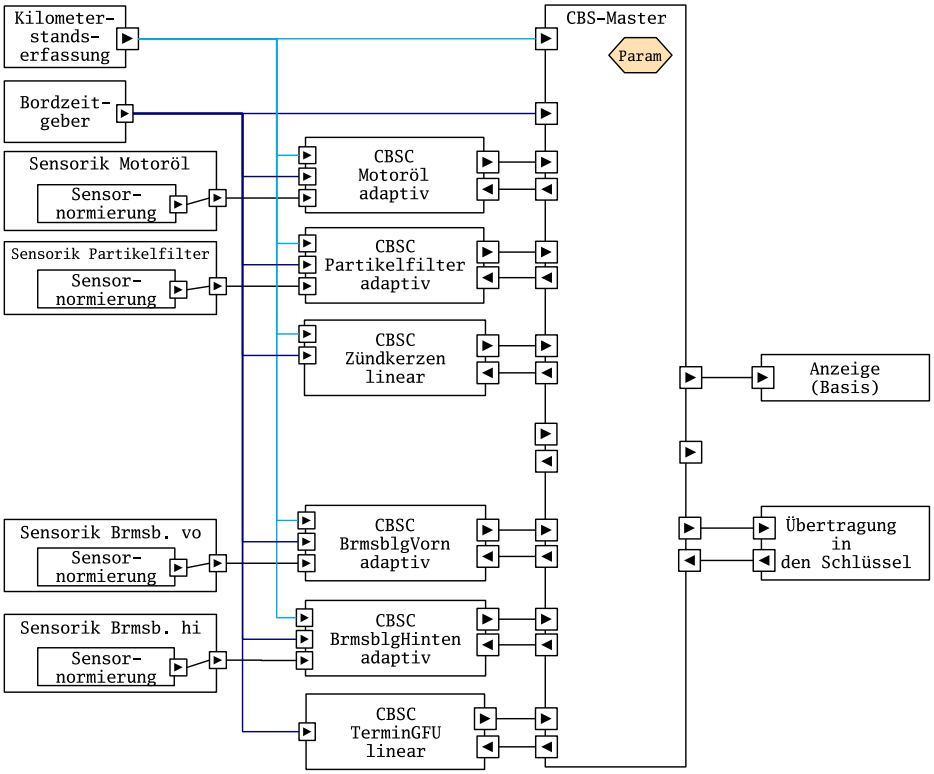


Abbildung 24 Softwarearchitektur: CBS-Softwarekomponentenproduktlinie (gesamt).

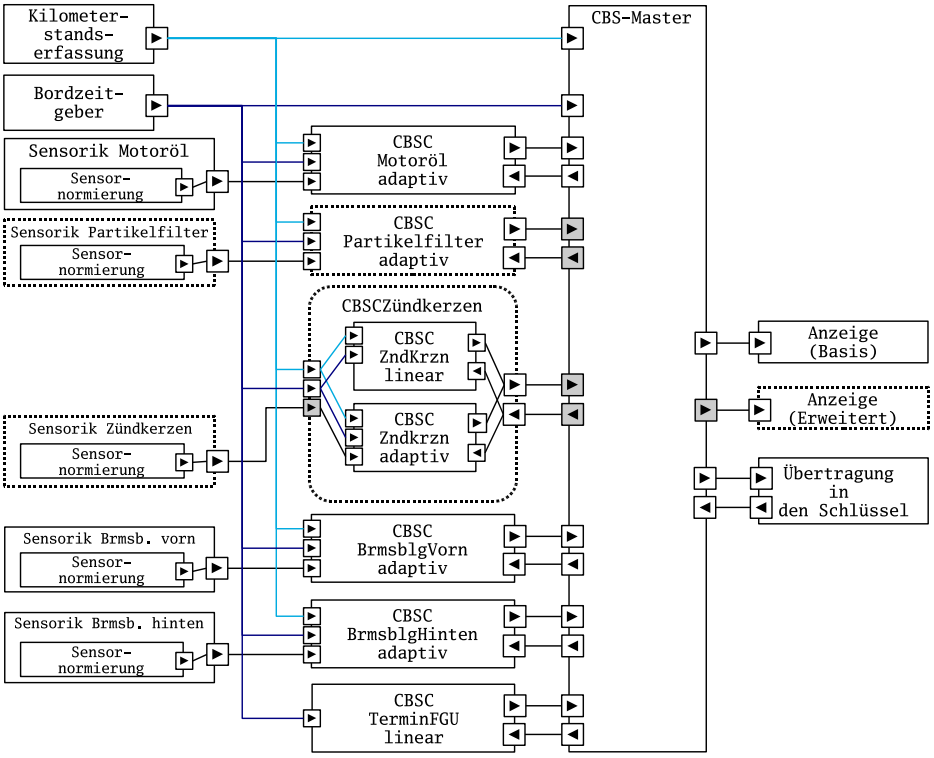


Abbildung 25 Softwarearchitektur: CBS-Softwarekomponentenproduktlinie (Master).

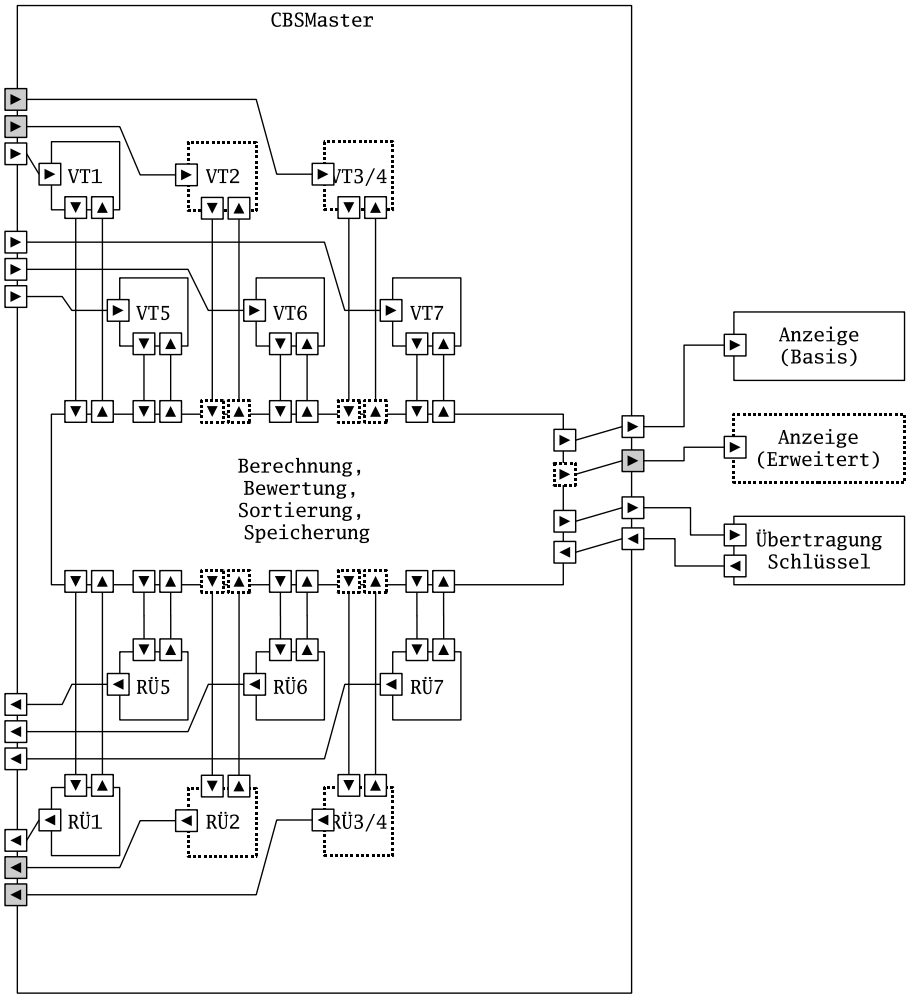


Abbildung 26 Softwarearchitektur: Produkt 1 aus CBS-Softwarekomponentenproduktlinie (gesamt).

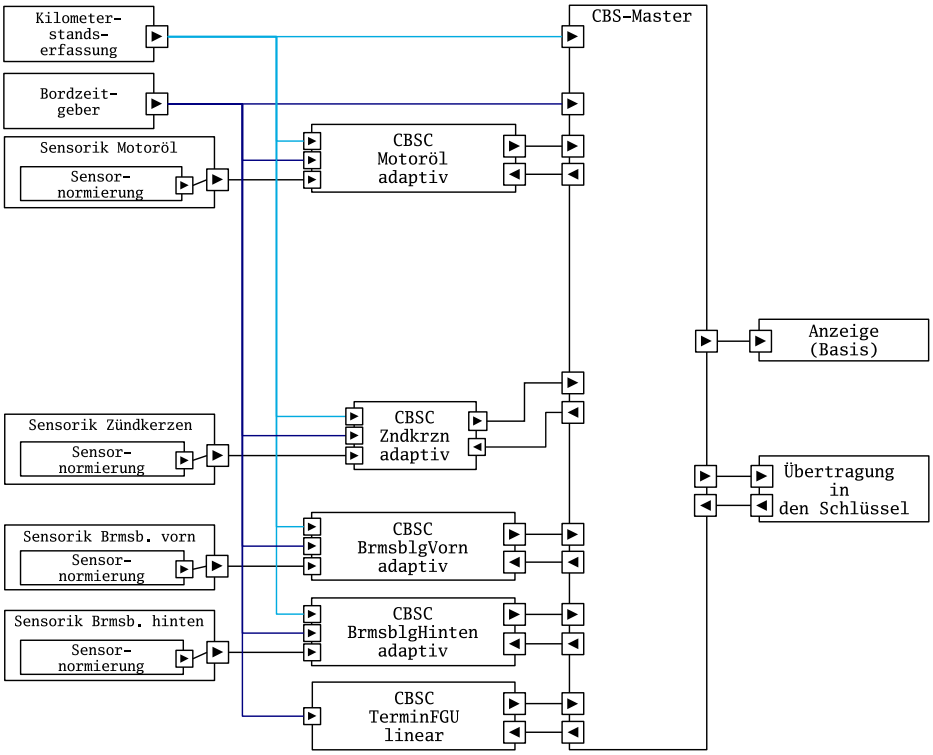


Abbildung 27 Softwarearchitektur: Produkt 1 aus CBS-Softwarekomponentenproduktlinie (Master).

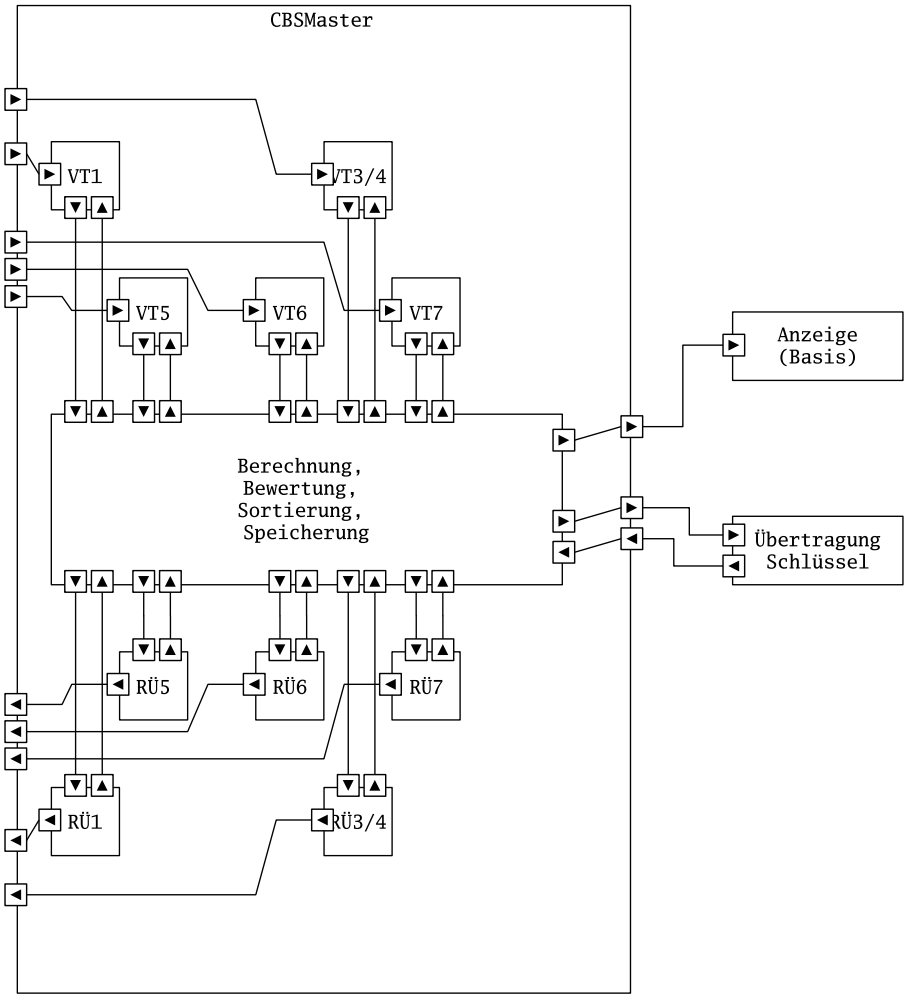


Abbildung 28 Softwarearchitektur: Produkt 2 aus CBS-Softwarekomponentenproduktlinie (gesamt).

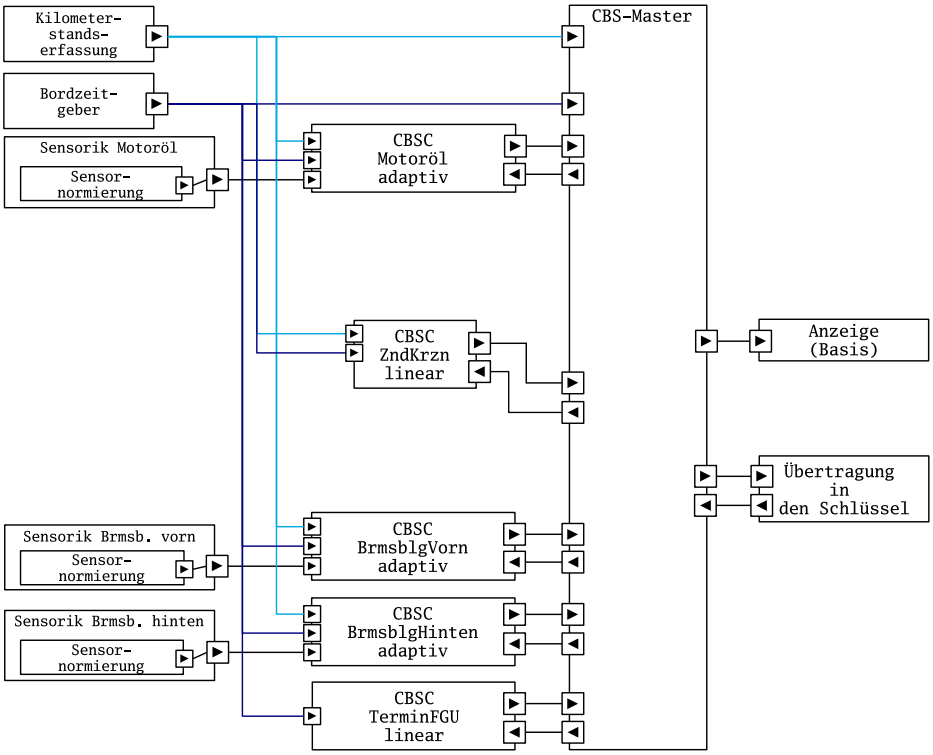


Abbildung 29 Softwarearchitektur: Produkt 2 aus CBS-Softwarekomponentenproduktlinie (Master).

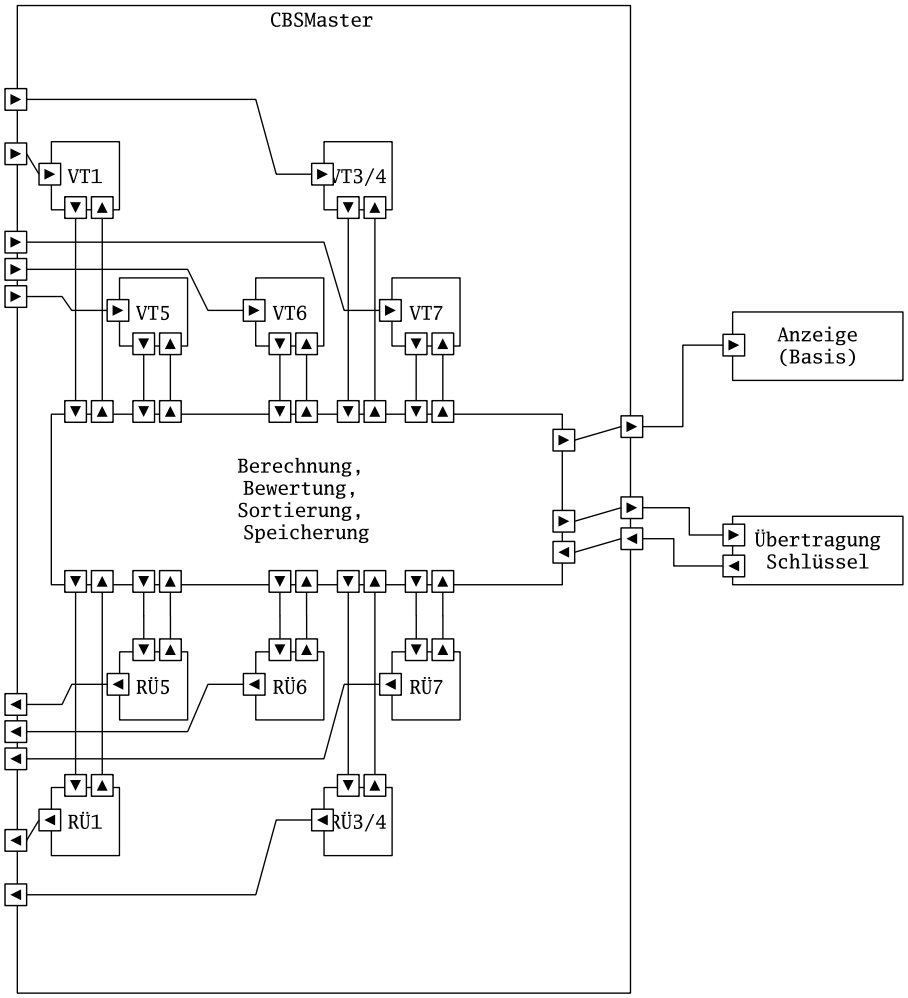


Abbildung 30 Softwarearchitektur: Produkt 3 aus CBS-Softwarekomponentenproduktlinie (gesamt).

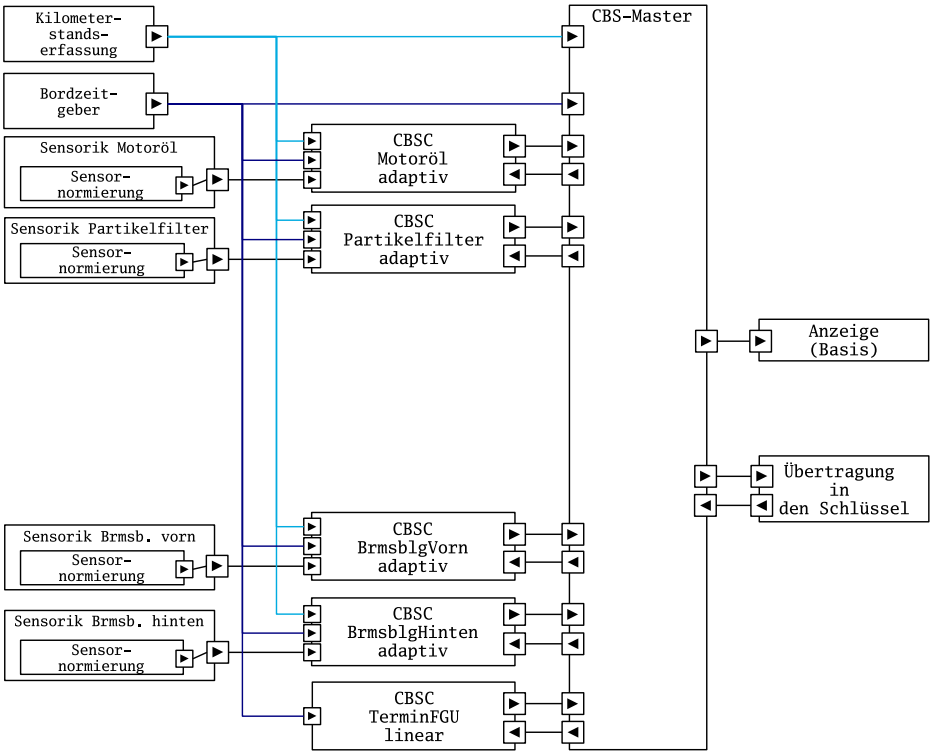
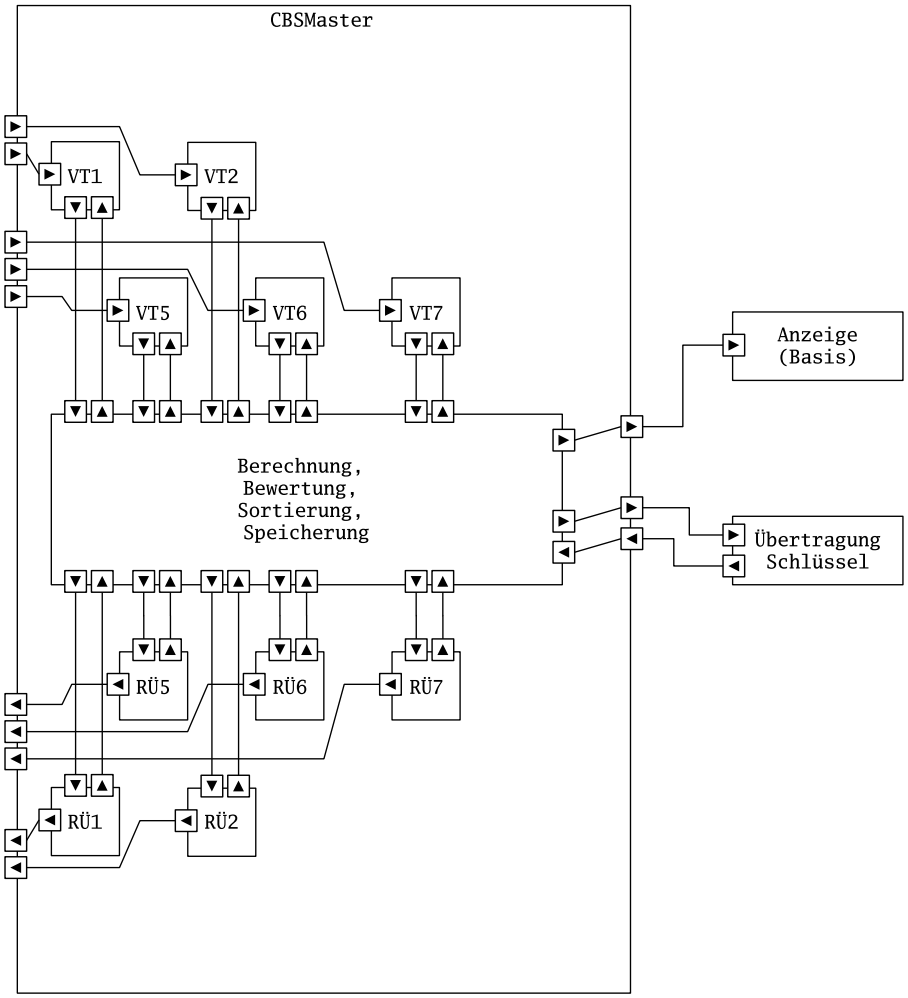


Abbildung 31 Softwarearchitektur: Produkt 3 aus CBS-Softwarekomponentenproduktlinie (Master).



6 Anwendung von Metriken auf die CBS-Modellierung

Im Folgenden werden die Ergebnisse der beispielhaften Anwendung der Metriken auf Basis von Prozessfamilienpunkten (PFP) [Kie06] auf den zuvor erstellten Modellierungen der logischen Architektur und der verschiedenen Softwarearchitekturen dargestellt. Die folgenden Ausführungen beziehen sich dabei auf die in [GKM07] geführte Untersuchung von Metriken für Produktlinien, insbesondere der PFP-Metriken (vgl. [GKM07, Abschnitt 4.3]).

Auf Basis der logischen Architektur, dessen Bewertung in Abschnitt 6.2 angegeben ist, wurden zwei konkurrierende Softwarearchitekturen erstellt, die ebenfalls beispielhaft mit den PFP-Metriken bewertet (Abschnitt 6.3.1 und Abschnitt 6.3.2) und dann miteinander verglichen wurden (Abschnitt 6.3.3). Ziel war, Aussagen dazu treffen zu können, welche der Softwarearchitekturen *besser* ist.

6.1 Produkte

Wie in Kapitel 3 beschrieben wurde, gehen wir im Beispiel von drei Produkten aus. Sie sind durch folgende Merkmale ausgezeichnet:

P1: Ottomotor, adaptive Zündkerzenverschleißerfassung, keine erweiterte Anzeige

P2: Ottomotor, lineare Zündkerzenverschleißerfassung, keine erweiterte Anzeige

P3: Dieselmotor, adaptive Partikelfilterverschleißerfassung, keine erweiterte Anzeige

Insgesamt gehen wir also von einer prinzipiellen Produkt(typ)anzahl von $PA = 3$ aus. Die Implementierungshäufigkeit setzen wir im Folgenden in Relation zur Produktanzahl. Für die Implementierungshäufigkeit (IH) der einzelnen Merkmale gehen wir beispielhaft von einer Gleichverteilung von Fahrzeugen mit Otto- und mit Dieselmotoren sowie von einer Gleichverteilung von Fahrzeugen mit adaptiver und mit linearer Zündkerzenverschleißerfassung aus. Unter Berücksichtigung der Tatsache, dass Zündkerzen nur bei Ottomotoren verbaut werden, kommen wir damit auf folgende Werte:

– Implementierungshäufigkeit für Ottomotoren: $IH_{Ottomotor} = 1/2 * PA$

- Implementierungshäufigkeit für Dieselmotoren: $IH_{Dieselmotor} = 1/2 * PA$
- Implementierungshäufigkeit für Funktionen bzw. Komponenten zur adaptiven Zündkerzenverschleißerfassung: $IH_{ZuendkerzenVE\ adaptiv} = 1/4 * PA$.
- Implementierungshäufigkeit für Funktionen bzw. Komponenten zur linearen Zündkerzenverschleißerfassung: $IH_{ZuendkerzenVE\ linear} = 1/4 * PA$.

In der Modellierung der logischen Architektur wurde die Funktionalität der Bremsbelagverschleißerfassung durch Wiederverwendung realisiert: die Bremsbelagverschleißerfassungsfunktion wird sowohl für die vorderen als auch die hinteren Bremsbeläge verwendet, d. h. in der logischen Architektur zweimal instanziiert. Damit ergibt sich, dass die zugehörigen Funktionen doppelt so häufig wie die Produktanzahl verwendet werden. In den Softwarearchitekturen des Fallbeispiels wurde solch eine Mehrfachverwendung von Komponenten derzeit noch nicht berücksichtigt.

6.2 Anwendung von Metriken auf das Modell der logischen Architektur von CBS

Die Anwendung der Metriken in diesem Abschnitt beziehen sich auf die Modellierung der logischen Architektur in Kapitel 4.

6.2.1 Kiebuschs Unjustierte PFP

Alle Funktion wurden als Echtzeitfunktionen mit weichen Echtzeitanforderungen (EGWs bzw. EVWs) bewertet. Optionale Funktionen und Funktionsalternativen wurden als horizontale EVWs betrachtet. Wenn die Funktionen optionale, abhängige oder variierende Ports haben, dann wurden sie als vertikale EGWs oder EVWs eingestuft. Zur Komplexitätsgewichtung wurden die Signale der Funktionsports gemäß der Portdeklarationen gezählt.

Zur Bewertung wurden alle atomaren Funktionen des Funktionsnetzes herangezogen, d. h. alle Blattknoten der Funktionshierarchie. Demzufolge mussten die in der Hierarchie enthaltenen strukturellen Varianzinformationen (XOR-Varianz und Optionalität bei hierarchisch zusammengesetzten Funktionen) auf die atomaren Funktionen entsprechend bezogen werden. Die in Tabelle 1 aufgelisteten atomaren Funktionen sind entsprechend ihrer hierarchischen Strukturierung sortiert.

Das Modellierungskonzept der Mehrfachverwendung von Funktionen (durch so

genannte »Function-Hooks«) ist im CBS-Funktionsnetz einmal angewendet worden: die Spezifikation der Verschleißbewertung für die vorderen und hinteren Bremsbeläge wurde durch die zweimalige Verwendung der wiederverwendbaren Funktion `CBSBremsbelag` umgesetzt. Dementsprechend positiv wirkt sich der ermittelte Umfang für diese wiederverwendbare Funktion bzgl. der Produktlinie aus.

Die gemäß der 3., 4. und 5. Metrik ermittelten Maße zur Bestimmung der unjustierten Prozessfamilienpunkte (UPFP) als abstraktes Umfangsmaß für die atomaren Funktionen des CBS-Funktionsnetzes sind in Tabelle 1 abgebildet.

Tabelle 1 UPFP für CBS-Funktionsnetz.

Funktion	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH ($\#Hooks$)	V	$UPFP$	$UPFP_{PA=3}$
`Bremsbelag Verschleiß`	7	3	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Bremsbelag Verfügbarkeit`	9	4	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Bremsbelag EinSer-viceIntervall`	9	4	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Bremsbelag Lebenszyklus`	9	5	h	23	–	1 PA (2)	–	11,5 / PA	3,83
`Motoröl Verschleiß`	3	1	m	15	–	1 PA	–	15 / PA	5
`Motoröl Verfügbarkeit`	9	4	h	23	–	1 PA	–	23 / PA	7,67
`Motoröl EinSer-viceIntervall`	9	4	h	23	–	1 PA	–	23 / PA	7,67
`Motoröl Lebenszyklus`	9	5	h	23	–	1 PA	–	23 / PA	7,67
`Zündkerzen linear Verfügbarkeit`	6	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen linear EinServiceInter-vall`	9	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen linear Lebenszyklus`	9	5	h	–	10	1/4 PA	–	40 / PA	13, 33
`Zündkerzen adaptiv Verschleiß`	3	1	m	–	7	1/4 PA	–	28 / PA	9,33
`Zündkerzen adaptiv Verfügbarkeit`	6	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen adaptiv EinServiceInter-vall`	9	4	h	–	10	1/4 PA	–	40 / PA	13,33
`Zündkerzen adaptiv Lebenszyklus`	9	5	h	–	10	1/4 PA	–	40 / PA	13, 33

(Weiter auf der nächsten Seite) ↪ . . .

Funktion	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH $(\#Hooks)$	V	$UPFP$	$UPFP_{PA=3}$
`Partikelfilter Verschleiß`	3	1	m	–	7	1/2 PA	–	14 / PA	4,67
`Partikelfilter adaptiv Verfügbarkeit`	9	4	h	–	10	1/2 PA	–	20 / PA	6,67
`Partikelfilter adaptiv EinServiceIntervall`	9	4	h	–	10	1/2 PA	–	20 / PA	6,67
`Partikelfilter adaptiv Lebenszyklus`	9	5	h	–	10	1/2 PA	–	20 / PA	6,67
`GFU Verfügbarkeit`	3	2	m	15	–	1 PA	–	15 / PA	5
`GFU EinServiceIntervall`	6	2	h	23	–	1 PA	–	23 / PA	7,67
`GFU Lebenszyklus`	6	5	h	23	–	1 PA	–	23 / PA	7,67
`Servicetermin Bestimmen`	12	2	h	23	10	1 PA	4/14	$\frac{135}{7\ PA}$ = 19,29/PA	6,43
`CBS Anzeige`	1	2	m	15	7	1 PA	1/3	$\frac{37}{3\ PA}$ = 12,33/PA	4,11
`CBS Serviceintervall Rücksetzen`	4	18	h	23	10	1 PA	12/22	$\frac{175}{11\ PA}$ = 15,91/PA	5,3
$\sum_f UPFP(f)$								ca. 580,53/PA	ca. 193,51

6.2.2 Kiebuschs 9. Metrik: Allgemeine Aufwandstreiber

Durch eine beispielhafte Bewertung der allgemeinen Aufwandstreiber gemäß der 9. Metrik kommen folgende Werte zustande:

- Dokumentation: 4
- Infrastruktur: 4
- Transitionsprozess: 2
- Wissensvermittlung: 0

$\Rightarrow\ SG^G = 7$

6.2.3 Kiebuschs 10. Metrik: Automotive-Aufwandstreiber

Durch beispielhafte Bewertung der Automotive-spezifischen Aufwandstreiber gemäß der 10. Metrik kommen folgende Werte zustande:

- Rechenleistung: 4
- Sicherheit: 4
- Speicherumfang: 2

$\Rightarrow$ $SG^D = 10$

6.2.4 Kiebuschs 12. Metrik: Justierte PFPs

Durch Bewertung der Aufwandstreiber ergibt sich gemäß der 12. Metrik folgender Justierungsfaktor:

- Justierungsfaktor: $J = 0,99$

Als Beispiel der Anwendung einer zählmodellspezifischen Justierung des ermittelten Umfangs sei im Folgenden die Metrik für ein neues Produkt angewendet:

- $\textit{justierte PFP}_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J = (\frac{580{,}53}{PA} + 0) * 0,99 = \frac{574{,}72}{PA}$
  (bei $PA = 3$: $191,57$)

6.2.5 Kiebuschs 13. Metrik

Diese Metrik wurde noch nicht angewendet.

6.2.6 Kiebuschs 14. Metrik

Gemäß der 14. Metrik und der oben ermittelten Maße würden folgende Aufwände nötig sein, um die anhand der logischen Architektur angegebene CBS-Produktlinie

zu implementieren:[6]

- Aufwandsprognose auf Basis unjustierter PFP:
  $2,4199\,PS * \frac{580{,}53}{PA} = \frac{1404{,}82\,PS}{PA} = \frac{175{,}6\,PT}{PA}$
  (bei $PA = 3$: $58,53\,PT$)

- Aufwandsprognose auf Basis justierter PFP:
  $2,0987\,PS * \frac{574{,}72}{PA} = \frac{1206{,}65\,PS}{PA} = \frac{150{,}77\,PT}{PA}$
  (bei $PA = 3$: $50,26\,PT$)

6.3 Anwendung von Metriken auf CBS-Softwarearchitekturen

Die Anwendung der Metriken in diesem Abschnitt beziehen sich auf die Modellierung der Softwarekomponentenarchitekturen in Kapitel 5.

6.3.1 Produktgenerische Produktliniensoftwarearchitektur für CBS (»CBS-150 %-Softwarearchitektur«)

Siehe hierzu die Modelle in Abschnitt 5.1.

Kiebuschs unjustierte PFP

Alle Softwarekomponenten wurden als Komponenten mit weichen Echtzeitanforderungen eingestuft. Außerdem wurden alle Softwarekomponenten den Gemeinsamkeiten (*EG*) der Produktlinie zugerechnet, da hier ein produktgenerischer Ansatz verfolgt wird. Zur Komplexitätseinstufung wurde nur die Anzahl der Eingangs- und Ausgangsports von Softwarekomponenten, jedoch nicht die Anzahl der Signale gezählt.

Zur Bewertung wurden alle atomaren Softwarekomponenten der Softwarearchitek-

[6] Gemäß der PFP-Metriken fließen in das Umfangsmaß sowohl Struktur- als auch Verhaltensinformationen ein. In der betrachteten Fallstudie sind die Funktionen jedoch noch nicht ausreichend mit Verhaltensmodellen hinterlegt worden, so dass sich der Aufwand derzeit nur aus der Struktur des Funktionsnetzes herleitet.
PS steht für die Maßeinheit Personenstunde, PT für Personentag.

tur herangezogen, d. h. alle Blattknoten der Hierarchie von Softwarekomponenten. Die produktgenerische Softwarearchitektur wurde so erstellt, dass sie keine explizite strukturelle Varianz enthält, d. h., die strukturelle Varianz der logischen Architektur wurde durch einen 150 %-Ansatz »aufgelöst«. Die in Tabelle 2 aufgelisteten atomaren Softwarekomponenten sind entsprechend ihrer hierarchischen Strukturierung in Sensor-, Client- und Masteranteile sortiert.

Die gemäß der 3., 4. und 5. Metrik ermittelten Maße zur Bestimmung der unjustierten Prozessfamilienpunkte (UPFP) als abstraktes Umfangsmaß für die atomaren Softwarekomponenten der produktgenerischen Softwarearchitektur sind in Tabelle 2 abgebildet.

Tabelle 2 Unjustierte PFP für produktgenerische Produktliniensoftwarearchitektur (CBS-150).

SW-Komponente	*#In*	*#Out*	*Kompl.*	UF_{gem}	UF_{var}	*IH*	*V*	*UPFP*	$UPFP_{PA=3}$
`Kilometerstands-erfassung`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Bordzeitgeber`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Motoröl`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Partikelfilter`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Zündkerzen`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Bremsbelag vorn`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`Sensorik Bremsbelag hinten`	0	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSC Motoröl adaptiv`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Partikelfilter adaptiv`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Zündkerzen linear`	3	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Zündkerzen adaptiv`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Bremsbelag vorn`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC Bremsbelag hinten`	4	1	m	15	–	1 PA	–	15 / PA	5
`CBSC TerminGFU linear`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT1`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT2`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT3`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT4`	2	1	g	10	–	1 PA	–	10 / PA	3,33
`CBSM VT5`	2	1	g	10	–	1 PA	–	10 / PA	3,33

(Weiter auf der nächsten Seite) ↪ . . .

SW-Komponente	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH	V	$UPFP$	$UPFP_{PA=3}$
CBSM VT6	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT7	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM Berechnen	15	17	h	23	–	1 PA	–	23 / PA	7,67
CBSM RÜ1	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ2	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ3	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ4	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ5	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ6	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ7	1	2	m	15	–	1 PA	–	15 / PA	5
Anzeige (Basis)	1	0	g	10	–	1 PA	–	10 / PA	3,33
Anzeige (Erweitert)	1	0	g	10	–	1 PA	–	10 / PA	3,33
Übertragung Schlüssel	1	1	g	10	–	1 PA	–	10 / PA	3,33
Σ_{swc}	64	53						398 / PA	132,67

Kiebuschs 12. Metrik: Justierte PFPs

Durch Bewertung der Aufwandstreiber (die von der Bewertung der logischen Architektur wiederverwendet wurde, siehe Abschnitt 6.2) ergibt sich gemäß der 12. Metrik folgender Justierungsfaktor:

– Justierungsfaktor: $J = 0,99$

Als Beispiel der Anwendung einer zählmodellspezifischen Justierung des ermittelten Umfangs sei im Folgenden die Metrik für ein neues Produkt angewendet:

– $\textit{justierte } PFP_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J = (\frac{398}{PA} + 0) * 0,99 = \frac{394,02}{PA}$
(bei $PA = 3$: $131,34$)

Kiebuschs 14. Metrik

Gemäß der 14. Metrik und der oben ermittelten Maße würden folgende Aufwände nötig sein, um die anhand der produktgenerischen Softwarearchitektur angegebene CBS-Produktlinie zu implementieren:[7]

[7] Es gelten hier die gleichen Einschränkungen wie bei der Berechnung des Umfangsmaßes für das Funktionsnetz, siehe Fußnote 6.

– Aufwandsprognose auf Basis unjustierter PFP:
$2,4199\,PS * \frac{398}{PA} = \frac{963{,}12\,PS}{PA} = \frac{120{,}39\,PT}{PA}$
(bei $PA = 3$: $40,13\,PT$)

– Aufwandsprognose auf Basis justierter PFP:
$2,0987\,PS * \frac{394{,}02}{PA} = \frac{826{,}93\,PS}{PA} = \frac{103{,}37\,PT}{PA}$
(bei $PA = 3$: $34,47\,PT$)

6.3.2 Produktspezifische Produktliniensoftwarearchitektur für CBS (»CBS-SWPL-Architektur«)

Siehe hierzu die Modelle in Abschnitt 5.2.

Kiebuschs unjustierte PFP

Alle Softwarekomponenten wurden als Komponenten mit weichen Echtzeitanforderungen eingestuft. Optionale Softwarekomponenten und Alternativen wurden als horizontale EVWs betrachtet. Wenn Softwarekomponenten optionale, abhängige oder variierende Ports besitzen, dann wurden sie als vertikale EGWs oder EVWs eingestuft. Zur Komplexitätseinstufung wurde nur die Anzahl der Eingangs- und Ausgangsports von Softwarekomponenten, jedoch nicht die Anzahl der Signale gezählt.

Zur Bewertung wurden alle atomaren Softwarekomponenten der Softwarearchitektur herangezogen, d. h. alle Blattknoten der Hierarchie von Softwarekomponenten. Die produktspezifische Softwarearchitektur wurde so erstellt, dass sie die strukturelle Varianz der logischen Architektur übernimmt. Demzufolge mussten die in der Hierarchie enthaltenen strukturellen Varianzinformationen (XOR-Varianz und Optionalität bei hierarchisch zusammengesetzten Softwarekomponenten) auf die atomaren Softwarekomponenten entsprechend bezogen werden. Die in Tabelle 3 aufgelisteten atomaren Softwarekomponenten sind entsprechend ihrer hierarchischen Strukturierung in Sensor-, Client- und Masteranteile sortiert.

Die gemäß der 3., 4. und 5. Metrik ermittelten Maße zur Bestimmung der unjustierten Prozessfamilienpunkte (UPFP) als abstraktes Umfangsmaß für die atomaren Softwarekomponenten der produktspezifischen Softwarearchitektur sind in Tabelle 3 abgebildet.

Tabelle 3 Unjustierte PFP für produktspezifische Produktliniensoftwarearchitektur (CBS-SWPL).

SW-Komponente	*#In*	*#Out*	*Kompl.*	UF_{gem}	UF_{var}	*IH*	*V*	*UPFP*	$UPFP_{PA=3}$
Kilometerstands-erfassung	0	1	g	10	–	1 PA	–	10 / PA	3,33
Bordzeitgeber	0	1	g	10	–	1 PA	–	10 / PA	3,33
Sensorik Motoröl	0	1	g	10	–	1 PA	–	10 / PA	3,33
Sensorik Partikelfilter	0	1	g	–	5	1/2 PA	–	10 / PA	3,33
Sensorik Zündkerzen	0	1	g	–	5	1/4 PA	–	20 / PA	6,67
Sensorik Bremsbelag vorn	0	1	g	10	–	1 PA	–	10 / PA	3,33
Sensorik Bremsbelag hinten	0	1	g	10	–	1 PA	–	10 / PA	3,33
CBSC Motoröl adaptiv	4	1	m	15	–	1 PA	–	15 / PA	5
CBSC Partikelfilter adaptiv	4	1	m	–	7	1/2 PA	–	14 / PA	4,67
CBSC Zündkerzen linear	3	1	m	–	7	1/4 PA	–	28 / PA	9,33
CBSC Zündkerzen adaptiv	4	1	m	–	7	1/4 PA	–	28 / PA	9,33
CBSC Bremsbelag vorn	4	1	m	15	–	1 PA	–	15 / PA	5
CBSC Bremsbelag hinten	4	1	m	15	–	1 PA	–	15 / PA	5
CBSC TerminGFU linear	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT1	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT2	2	1	g	–	5	1/2 PA	–	10 / PA	3,33
CBSM VT3/4	2	1	g	–	5	1/2 PA	–	10 / PA	3,33
CBSM VT5	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT6	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM VT7	2	1	g	10	–	1 PA	–	10 / PA	3,33
CBSM Berechnen	13	15	h	23	10	1 PA	$\frac{9}{28}$	$\frac{527}{28\ \mathrm{PA}}$	6,27
CBSM RÜ1	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ2	1	2	m	–	7	1/2 PA	–	14 / PA	4,67
CBSM RÜ3/4	1	2	m	–	7	1/2 PA	–	14 / PA	4,67
CBSM RÜ5	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ6	1	2	m	15	–	1 PA	–	15 / PA	5
CBSM RÜ7	1	2	m	15	–	1 PA	–	15 / PA	5
Anzeige (Basis)	1	0	g	10	–	1 PA	–	10 / PA	3,33

(Weiter auf der nächsten Seite) ↪ . . .

SW-Komponente	$\#In$	$\#Out$	$Kompl.$	UF_{gem}	UF_{var}	IH	V	$UPFP$	$UPFP_{PA=3}$
`Anzeige (Erweitert)`	1	0	g	–	5	1 PA[8]	–	5 / PA	1,67
`Übertragung Schlüssel`	1	1	g	10	–	1 PA	–	10 / PA	3,33
Σ_{swc}	59	48						$\frac{11111}{28\ \text{PA}}$	132,27
								396,82/PA	

Kiebuschs 12. Metrik: Justierte PFPs

Durch Bewertung der Aufwandstreiber (die von der Bewertung der logischen Architektur wiederverwendet wurde, siehe Abschnitt 6.2) ergibt sich gemäß der 12. Metrik folgender Justierungsfaktor:

– Justierungsfaktor: $J = 0,99$

Als Beispiel der Anwendung einer zählmodellspezifischen Justierung des ermittelten Umfangs sei im Folgenden die Metrik für ein neues Produkt angewendet:

– $justierte\ PFP_{\substack{Produkt \\ (neu)}} = (NPFP + MPFP) * J = (\frac{11111}{28PA} + 0) * 0,99 = \frac{392,85}{PA}$
(bei $PA = 3$: $130,95$)

Kiebuschs 14. Metrik

Gemäß der 14. Metrik und der oben ermittelten Maße würden folgende Aufwände nötig sein, um die anhand der produktspezifischen Softwarearchitektur angegebene CBS-Produktlinie zu implementieren:[9]

– Aufwandsprognose auf Basis unjustierter PFP:
$2,4199\,PS * \frac{11111}{28\,PA} = \frac{960,27\,PS}{PA} = \frac{120,03\,PT}{PA}$
(bei $PA = 3$: $40,01\,PT$)

[8] In der Fallstudie wird die erweiterte Anzeige in keinem der Produkte verwendet. Es ist in den Metriken nicht vorgesehen, Varianten zu planen, die sich nicht in einem Produkt widerspiegeln (es gäbe eine Division durch Null). Da jede vorgesehene Variante mindestens einmal implementiert werden muss, wird hier die Implementierungshäufigkeit mit $1PA$ angenommen.

[9] Es gelten hier die gleichen Einschränkungen wie bei der Berechnung des Umfangsmaßes für das Funktionsnetz, siehe Fußnote 6.

– Aufwandsprognose auf Basis justierter PFP:
$2,0987\,PS * \frac{392{,}85}{PA} = \frac{824{,}48\,PS}{PA} = \frac{103{,}06\,PT}{PA}$
(bei $PA = 3$: $34,35\,PT$)

6.3.3 Vergleich der Softwarearchitekturen

In Tabelle 4 ist eine Übersicht zum Vergleich der produktgenerischen Produktlinien-softwarearchitektur (Spalte »CBS-SW150«) mit der produktspezifischen Produktlinien-softwarearchitektur (Spalte »CBS-SWPL«) angegeben.

In den Spalten »CBS-SWPL: P1/2/3« der Tabelle sind die Kennzahlen für die Einzelprodukte P1, P2 und P3 auf Basis der produktgenerischen Softwarearchitektur angegeben. Die Zeilen »# XXX« der Tabelle geben die Anzahl der jeweils modellierten Elemente in den Softwarearchitekturmodellen wieder und stellen somit die in den Modellen enthaltenen Strukturinformationen dar, die einfach gezählt werden können.

Die beiden unteren Zeilen der Tabelle geben die unjustierten Prozessfamilienpunkte für die beiden Produktlinienarchitekturen bzw. die drei spezifischen Produktarchitekturen an. Die Zeile »UPFP als Indiz für den Aufwand« interpretiert die Werte als Umfangsmaß, die zur Schätzung des Entwicklungsaufwands gemäß der weiteren PFP-Metriken verwendet werden. Dies entspricht der intendierten Verwendung der PFP-Metriken. Die Zeile »UPFP als Indiz für die SW-Größe« interpretiert die UPFP-Werte als Indiz für die Codegröße der zu implementierenden Softwarekomponenten für die Produkte auf Basis des jeweiligen Lösungsentwurfs. Diese Interpretation ist in den Arbeiten zu den PFP-Metriken nicht explizit vorgeschlagen, bietet sich aber im vorliegenden Anwendungskontext an.

Tabelle 4 Vergleich der Softwarearchitekturen.

	CBS-SW150	**CBS-SWPL**	**CBS-SWPL: P1**	**CBS-SWPL: P2**	**CBS-SWPL: P3**
# **Komponenten**	32	30	24	23	24
# **Eingangsports**	64	59	46	45	46
# **Ausgangsports**	53	48	39	38	39
# **Ports**	117	107	85	83	85
# **Komponenten CBS-Clients**	14	14	11	10	11
# **Eingangsports CBS-Clients**	25	25	18	17	18

(Weiter auf der nächsten Seite) ↪ . . .

	CBS-SW150	CBS-SWPL	CBS-SWPL: P1	CBS-SWPL: P2	CBS-SWPL: P3
# **Ausgangsports CBS-Clients**	14	14	11	10	11
# **Komponenten CBS-Master**	15	13	11	11	11
# **Eingangsports CBS-Master**	36	31	26	26	26
# **Ausgangsports CBS-Master**	38	33	27	27	27
UPFP als Indiz für den Aufwand	398 / PA bei PA=3: 132,67	396,82 / PA bei PA=3: 132,27	298	288	298
UPFP als Indiz für die SW-Größe	P1: 328 P2: 318 P3: 328	–	298	288	298

Auf Basis der mit Hilfe der PFP-Metriken ermittelten Werte lassen sich zunächst folgende Alternativen bewerten:

1 Vergleich unterschiedlicher Lösungsentwürfe bei der Produktlinienentwicklung (Dieser Vergleich entspricht der in [GKM07] diskutierten Bewertungsaufgabe.)

2 Einzelentwicklung von Produkten gegenüber einer Produktlinienentwicklung (Dieser Vergleich wird in der Fachliteratur zur Softwareproduktlinientechnik überlicherweise angestellt.)

Für den Vergleich können zusätzlich folgende Kriterien und Interpretationen des Umfangsmaßes hinzugezogen werden:

– Schätzung des Entwicklungsaufwands,
– Schätzung der Codegröße der implementierten Softwarekomponenten.

Die beiden Softwarearchitekturmodelle repräsentieren bisher nur eine minimale Produktlinie: in der Produktlinienbeschreibung wurden zwei Varianzpunkte (Otto vs. Diesel sowie lineare vs. adaptive Zündkerzenverschleißerfassung) betrachtet. Darauf aufbauend wurden drei Produkte für die Produktlinie definiert. Daher unterscheiden sich die beiden Produktliniensoftwarearchitekturen nicht sehr in den entsprechenden Maßen, wie in der Tabelle ersichtlich ist. Nichtsdestotrotz lassen sich einige generelle Aussagen ableiten, die durch weitere Fallstudien zu überprüfen sind.

Vergleich der beiden Lösungsentwürfe bei der Produktlinienentwicklung bzgl. Entwicklungsaufwand

Werden die strukturellen Merkmale der beiden Architekturen (Anzahl der atomaren Softwarekomponenten, Anzahl der Ports) miteinander verglichen, so fällt die produktgenerische Architektur etwas größer aus. Am deutlichsten ist der Unterschied für die Masteranteile: der produktgenerische `CBS-Master` ist für sämtliche potenziell in der Produktlinie vorgesehenen Verschleißumfänge und ihren Varianten ausgelegt, während der `CBS-Master` der produktspezifischen Architektur mit den Varianten bei den Umfängen besser skalierbar ist.

Bei Betrachtung der unjustierten Prozessfamilienpunkte beider Architekturen als Indiz für den Entwicklungsaufwand schneidet die produktgenerische Architektur geringfügig schlechter ab. Zwar werden sämtliche Komponenten für alle Produkte verwendet, was sich im Allgemeinen günstig auf den Aufwand auswirkt, jedoch fallen die Softwarekomponenten etwas komplexer aus, weil die Varianz in die Programmlogik verschoben ist (bspw. durch auszuwertende Konfigurationsparameter beim Systemstart bzw. zur Ausführungszeit; als strukturelles Indiz hierfür gelten die höhere Anzahl an Komponenten sowie die höhere Anzahl an Ports, die eine höhere Komplexitätseinstufung der Komponenten erwirken).

Bei der produktspezifischen Architektur können die Komponenten weniger komplex in der Programmlogik ausgelegt werden (als Indiz hierfür gelten die etwas niedrigere Anzahl von Eingangs- und Ausgangsports der Softwarekomponenten). Der Aufwand in der Generierung solcher Komponenten sowie ihre Verwaltung könnte jedoch auch aufwandserhöhend sein, dies müsste unter Umständen durch Erfahrungswerte bei der Berechnung der UPFPs sowie in der Justierung des Umfangsmaßes einfließen.

Da die betrachtete Produktlinie in der Fallstudie klein ist, fallen die Unterschiede in den UPFPs relativ gering aus. Hier sind noch weitere unterschiedliche Fallstudien nötig, um Erfahrungen zu sammeln, wie sich ein unterschiedlicher Varianzgrad oder die Anzahl der Produkte auf die Bewertung der beiden Softwarearchitekturen auswirkt.

Vergleich der Einzelentwicklung von Produkten mit einer Produktlinienentwicklung

In Tabelle 4 sind die strukturellen Maße als auch die UPFPs für die Einzelprodukte, die idealerweise auf Basis der produktspezifischen Produktliniensoftwarearchitektur erzeugt werden, angegeben. Diese Zahlen können so interpretiert werden, als seien die drei Produkte jeweils in einem eigenständigen Entwicklungsprojekt entstanden. Hiermit lässt sich die in der Produktlinienliteratur häufig gestellte Frage,

ob ein Unternehmen seine Produkte in Einzelentwicklungen oder auf Basis eines Produktlinienansatzes erstellt, relativ eindeutig beantworten: Der Umfang von drei Einzelentwicklungen von insgesamt 884 UPFP im Vergleich zum Umfang auf Basis der Produktlinienentwicklung von 398 $UPFP_{CBS\text{-}150}$ bzw. 396,82 $UPFP_{CBS\text{-}SWPL}$ ist mehr als das Doppelte, bereits bei dieser kleinen Fallstudie. Auch der relative Umfang für ein Produkt fällt auf Basis einer Produktlinie besser aus: 132,67 $UPFP_{CBS\text{-}150}$ bzw. 132,27 $UPFP_{CBS\text{-}SWPL}$ zu 298 $UPFP_{P1,\ P3}$ bzw. 288 $UPFP_{P2}$.

Vergleich der beiden Lösungsentwürfe bei der Produktlinienentwicklung bzgl. Codegröße

Ein weiteres Bewertungskriterium, das in der Automotive-E/E-Entwicklung eine große Bedeutung hat, ist die zu erwartende Codegröße der Softwarekomponenten. Hier kommt zum Tragen, dass einerseits genug Speicher seitens der Hardware im Fahrzeug zur Verfügung stehen muss, um den ausführbaren Softwarebinärcode im RAM, ROM oder Flashspeicher abzulegen, anderseits auch die Flashzeiten, d. h. die Dauer des »Hochladens« der Software ins Fahrzeug während oder nach der Montage des Fahrzeugs, eingeplant werden müssen.

Da das Umfangsmaß sich an strukturellen Eigenschaften der modellierten Softwarekomponenten orientiert, können unjustierte Prozessfamilienpunkte auch als Indiz für die zu erwartende Codegröße der Softwarekomponenten (des Binärcodes bzw. der Speicherbedarf zur Laufzeit) verwendet werden.[10] Diese Interpretation ist in der Zeile »UPFP als SW-Größe« in Tabelle 4 angegeben. Hierzu mussten die UPFPs für jedes Produkt – sowohl auf Basis des produktgenerischen als auch des produktspezifischen Ansatzes – ermittelt werden.

Die UPFP-Werte für die Einzelprodukte auf Basis des produktgenerischen Ansatzes (328 bzw. 318 UPFP) weichen vom UPFP-Wert der generischen Produktlinienarchitektur (398 UPFP) ab, weil sich die Produkte auf Infrastrukturebene unterscheiden: Fahrzeuge mit Ottomotoren besitzen nicht das DDE-Steuergerät, auf dem die Verschleißerfassung für den Partikelfilter installiert worden wäre. Demzufolge fallen die Umfangsmaße für Einzelprodukte in der Interpretation als Indiz für die Codegröße kleiner aus.

[10] Diesem Ansatz liegt folgende Annahme zu Grunde: Da für die UPFPs die Anzahl der Eingangs- und Ausgangssignale (bzw. Anzahl der Kanten und Knoten in den Statecharts) berücksichtigt wird, kann davon ausgegangen werden, dass die Größe des Softwarecodes sich proportional zur Anzahl der verarbeiteten Signale (bzw. Anzahl der Kanten und Knoten) verhält. Analoges gilt auch für den Speicherbedarf zur Laufzeit.

Beim Vergleich des produktgenerischen und des produktspezifischen Ansatzes zeigt sich, dass sämtliche Produkte auf Basis des produktspezifischen Ansatzes kleiner als die Produkte auf Basis des produktgenerischen Ansatzes ausfallen: z. B. 298 $UPFP_{P1(CBS\text{-}SWPL)}$ zu 328 $UPFP_{P1(CBS\text{-}150)}$. Der Grund hierfür liegt darin, dass kein »toter« Code für Varianten, die nicht zum Zuge kommen, ins Fahrzeug installiert wird. Auch der Speicherbedarf zum Ausführungszeitpunkt ist niedriger, weil weniger CBS-Umfänge vom `CBS-Master` auszuwerten sind.

Resümee

Zusammenfassend lässt sich sagen, dass auf Basis der in der CBS-Fallstudie erfassten Informationen die Lösungsalternative für die Softwareentwicklung auf Basis des produktspezifischen Ansatzes besser als die Lösungsalternative auf Basis des produktgenerischen Ansatzes erscheint. Einschränkend muss aber gesagt werden, dass nicht alle Bewertungskriterien in der oben angestellten Bewertung berücksichtigt wurden. Einige Faktoren, z. B. eine quantifizierte Bewertung der Verwendung der jeweils notwendigen Softwaretechnologien mit ihren Konsequenzen, wurden ebensowenig explizit berücksichtigt. Auch die Feinjustierung der PFP-Metriken konnte noch nicht erfolgen. Bisher wurden nur die vorgeschlagenen Grenzwerte und Erfahrungswerte aus [Kie06] verwendet.

7 Zusammenfassung

In diesem Bericht wurde die Bearbeitung der Fallstudie »Condition-Based Service« im Rahmen des Projekts VEIA fortgeführt. Ziel der Modellierung war die Anwendung von Metriken für die Architekturbewertung von Produktlinien. Dies ist in [GKM07] beschrieben. Die hier vorgestellte Modellierung diente als Fallstudie für die Anwendung von Metriken, so dass primär Wert darauf gelegt wurde, die charakteristischen Varianzkonzepte für die Darstellung von Produktlinien in den VEIA-Entwicklungsartefakten anzuwenden, als eine vollständige Beschreibung der Fallstudie CBS zu erreichen. Die Modelle repräsentieren einen beispielhaften Auszug aus der heutzutage in Fahrzeugen realisierten CBS-Funktionalität.

Literatur

[GEKM07] *Große-Rhode*, Martin ; *Euringer*, Simon ; *Kleinod*, Ekkart ; *Mann*, Stefan: Grobentwurf des VEIA-Referenzprozesses / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Januar 2007. – ISST-Bericht 80/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[GKM07] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Entscheidungsgrundlagen für die Entwicklung von Softwareproduktlinien / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Oktober 2007. – ISST-Bericht 83/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[Kie06] *Kiebusch*, Sebastian: *Metriken für prozessorientierte Software-System-Familien: Umfangskalkulation sowie Aufwandsprognose im Electronic Business und Automobilbereich*. Leipzig, Germany, Institut für Wirtschaftsinformatik, Universität Leipzig, Dissertation, 2006. http://www.kiebusch.de/